Access DB# 112930

# SEARCH REQUEST FORM

**Scientific and Technical Information Center**

Requester's Full Name: Qamrun Nahar Examiner #: 79621 Date: 1/28/04
Art Unit: 2124 Phone Number 305-7699 Serial Number: 09/747824
Mail Box and Bldg/Room Location: PK2-5B16 Results Format Preferred (circle): PAPER DISK E-MAIL

**If more than one search is submitted, please prioritize searches in order of need.**
**********

Please provide a detailed statement of the search topic, and describe as specifically as possible the subject matter to be searched. Include the elected species or structures, keywords, synonyms, acronyms, and registry numbers, and combine with the concept or utility of the invention. Define any terms that may have a special meaning. Give examples or relevant citations, authors, etc, if known. Please attach a copy of the cover sheet, pertinent claims, and abstract.

Title of Invention: Assembly language Code Compilation for an instruction-set architecture containing new instructions using the prior assembler
Inventors (please provide full names): John Simons

Earliest Priority Filing Date: 12/22/00

***For Sequence Searches Only* Please include all pertinent information (parent, child, divisional, or issued patent numbers) along with the appropriate serial number.**

Assignee: Hitachi America, Ltd.
Inventor Name: "Spoof Assembler"
I need conference papers by this Assignee and Inventor regarding an invention called "Spoof. Assembler" between the dates 1994 to 2000.
Goal: I need high-level information/documentation. I need documentation that this invention was made public. See the attached "Statement of Facts" submitted by Applicant.

**********

**STAFF USE ONLY**

Searcher: Teresa Esterfield
Searcher Phone #: 308-7795
Searcher Location: 4B30
Date Searcher Picked Up: 1/29/04 9:46am
Date Completed: 1/29/04 3 36pm
Searcher Prep & Review Time: _____
Clerical Prep Time: _____
Online Time: _____

**Type of Search**

NA Sequence (#) _____
AA Sequence (#) _____
Structure (#) _____
Bibliographic _____
Litigation _____
Fulltext _____
Patent Family _____
Other _____

**Vendors and cost where applicable**

STN _____
Dialog _____
Questel/Orbit _____
Dr.Link _____
Lexis/Nexis _____
Sequence Systems _____
WWW/Internet _____
Other (specify) _____

PTO-1590 (8-01)

# STIC Search Report
## EIC 2100

STIC Database Tracking Number: 112930

| TO: | From: |
|---|---|
| Qamrun Nahar<br>Location: 5B46<br>Art Unit : 2124<br>Thursday, January 29, 2004<br><br>Case Serial Number: 09747824 | Terese Esterheld<br>Location: EIC 2100<br>PK2-4B30<br>Phone: 308-7795<br><br>Terese.esterheld@uspto.gov |

## Search Notes

Dear Examiner Nahar,

Attached, please find the results of your search request for application 09747824. I have concentrated on finding information on the inventor or Hitachi and Spoof Assembler. Also searched Spoof Assembler.

Since, I did not find anything, I did an inventor search in the Foreign Patents. There I found two versions of the application. Also located additional information in order to prepare an additional search strategy. I did find a few hits that may be of value to you.

Look over the complete set of hits as there items not marked that may also be of help.

Please let me if you need additional information on this search.

Thank you for coming to EIC 2100.

Terese Esterheld

```
Set     Items   Description
S1        176   AU=(SIMONS, J? OR SIMONS J?)
S2         16   S1 AND IC=G06F?
S3          1   S1 AND ASSEMBLY()LANGUAGE
S4         16   S2 OR S3
File 347:JAPIO Oct 1976-2003/Sep(Updated 040105)
         (c) 2004 JPO & JAPIO
File 348:EUROPEAN PATENTS 1978-2004/Jan W04
         (c) 2004 European Patent Office
File 349:PCT FULLTEXT 1979-2002/UB=20040122,UT=20040115
         (c) 2004 WIPO/Univentio
File 350:Derwent WPIX 1963-2004/UD,UM &UP=200407
         (c) 2004  Thomson Derwent
```

4/5/1 (Item 1 from file: 347)
DIALOG(R)File 347:JAPIO


07328449 **Image available**
METHOD FOR COMPILING **ASSEMBLY LANGUAGE** CODE FOR INSTRUCTION SET ARCHITECTURE INCLUDING NEW INSTRUCTION USING CONVENTIONAL ASSEMBLER

PUB. NO.: 2002-196937 [JP 2002196937 A]
PUBLISHED: July 12, 2002 (20020712)
INVENTOR(s): **SIMONS JOHN**
APPLICANT(s): HITACHI LTD
APPL. NO.: 2001-328956 [JP 20011328956]
FILED: October 26, 2001 (20011026)
PRIORITY: 00 747824 [US 2000747824], US (United States of America), December 22, 2000 (20001222)
INTL CLASS: **G06F-009/45**

ABSTRACT

PROBLEM TO BE SOLVED: To provide a simple and effective method for compiling an instruction of a new **assembly language** instruction set architecture using a previous assembler.

SOLUTION: An assembler expansion instruction set architecture ISA is formed from the newest ISA to which a new instruction is added. The assembly of a source code output in hybrid of present and new **assembly language** instructions is attained by preprocessing of the source code in order to generate a temporary file 46 including a previous instruction 40a and data designation 41 to each of new assembly instructions having object code equivalence of the new instruction 40b as a data independent function, Thereafter, the temporary file 46 is used for the previous assembler 40a in order to generate object codes corresponding to each of the previous **assembly language** instructions. Resultantly, an executable machine language program for the new ISA is generated after linking.

4/5/7 (Item 1 from file: 350)
DIALOG(R)File 350:Derwent WPIX


015692813 **Image available**
WPI Acc No: 2003-755002/200371
XRPX Acc No: N03-604956

**Source code conversion in e.g. microcomputer, involves applying instructions of temporary file to assembler to produce object code corresponding to old instructions, and data forming object code for new instructions**

Patent Assignee: HITACHI LTD (HITA ); HITACHI AMERICA LTD (HITA )
Inventor: **SIMONS J**
Number of Countries: 002 Number of Patents: 002
Patent Family:

| Patent No | Kind | Date | Applicat No | Kind | Date | Week |
|---|---|---|---|---|---|---|
| US 20020083421 | A1 | 20020627 | US 2000747824 | A | 20001222 | 200371 B |
| JP 2002196937 | A | 20020712 | JP 2001328956 | A | 20011026 | 200371 |

Priority Applications (No Type Date): US 2000747824 A 20001222
Patent Details:

| Patent No | Kind | Lan | Pg | Main IPC | Filing Notes |
|---|---|---|---|---|---|
| US 20020083421 | A1 | | 8 | G06F-009/45 | |
| JP 2002196937 | A | | 7 | G06F-009/45 | |

Abstract (Basic): US 20020083421 A1

NOVELTY - The method involves copying several instructions to a temporary file (46). The instructions of the temporary file are applied to an assembler to produce an object code corresponding to the old instructions (40a,40b), and a data forming object code for new instructions.

DETAILED DESCRIPTION - An INDEPENDENT CLAIM is also included for a source code assembling method.

USE - For converting source code to object code in microcomputers and microprocessors.

ADVANTAGE - Enables assembling code for new instruction-set architecture (ISA) using older assembler such that development of extended ISA can continue concomitant with the development of new assembler.

DESCRIPTION OF DRAWING(S) - The figure shows a schematic view explaining source code conversion.

source file (40)
old assembly instructions (40a,40b)
preprocessor (42)
temporary source code file (46)
object file (50)
new processor (58)
pp; 8 DwgNo 3/4

Title Terms: SOURCE; CODE; CONVERT; MICROCOMPUTER; APPLY; INSTRUCTION; TEMPORARY; FILE; ASSEMBLE; PRODUCE; OBJECT; CODE; CORRESPOND; INSTRUCTION ; DATA; FORMING; OBJECT; CODE; NEW; INSTRUCTION
Derwent Class: T01
International Patent Class (Main): **G06F-009/45**
File Segment: EPI

4/5/14 (Item 8 from file: 350)
DIALOG(R)File 350:Derwent WPIX


011025664 **Image available**
WPI Acc No: 1997-003588/199701
XRPX Acc No: N97-003171

**Automatic debugging command file formation for computer system - by re-assembling corrected source program counter which includes implanting debugging command at new opposed-position in automatically generating new command file with new brake point command**

Patent Assignee: HITACHI LTD (HITA ); HITACHI AMERICA LTD (HITA )
Inventor: SHRIDHAR A; **SIMONS J**

Number of Countries: 002 Number of Patents: 002
Patent Family:

| Patent No | Kind | Date | Applicat No | Kind | Date | Week | |
|---|---|---|---|---|---|---|---|
| JP 8272648 | A | 19961018 | JP 95285382 | A | 19951101 | 199701 | B |
| US 5815714 | A | 19980929 | US 94366050 | A | 19941229 | 199846 | |
| | | | US 97839229 | A | 19970421 | | |

Priority Applications (No Type Date): US 94366050 A 19941229; US 97839229 A 19970421
Patent Details:

| Patent No | Kind | Lan | Pg | Main IPC | Filing Notes |
|---|---|---|---|---|---|
| JP 8272648 | A | | 13 | G06F-011/28 | |
| US 5815714 | A | | | G06F-009/45 | Cont of application US 94366050 |

Abstract (Basic): JP 8272648 A

The method involves embedding one debugging command to one line of a source program counter. A brake point command is formed corresp. to each extracted implanting debugging command. A command file is generated by writing the brake point command and the debugging command to a debugging command file. An object code is formed from the source program command which is retouched by removing the brake point command and the implanting debugging command from the debugging command file.

A new command file which includes the new brake point command corresp. to the implanting debugging command, is generated. The implanting debugging command is in a new opposed-position on the source program counter during re-assembling of the corrected source program counter.

ADVANTAGE - Enables automatic formation of debugging command file utilised by debugger during simulation execution of object code drawn from source program counter.

Dwg.2/8

Title Terms: AUTOMATIC; DEBUG; COMMAND; FILE; FORMATION; COMPUTER; SYSTEM; ASSEMBLE; CORRECT; SOURCE; PROGRAM; COUNTER; IMPLANT; DEBUG; COMMAND; NEW ; OPPOSED; POSITION; AUTOMATIC; GENERATE; NEW; COMMAND; FILE; NEW; BRAKE; POINT; COMMAND
Derwent Class: T01
International Patent Class (Main): **G06F-009/45 ; G06F-011/28**
File Segment: EPI

```
Set     Items   Description
S1       2939   AU=(SIMONS, J? OR SIMONS J?)
S2       2633   S1 NOT PY>2000
S3        882   S2 NOT PY<1994
S4          0   S3 AND (ASSEMBLER? OR LINKER? OR COMPILER? OR LANGUAGE()PR-
              OCESSOR)
S5          1   S3 AND (OBJECT OR MACHINE()(CODE OR CODES) OR ASSEMBLY()LA-
              NGUAGE()SOURCE()CODE OR ASSEMBLY()LANGUAGE()INSTRUCTION)
S6          2   S3 AND (SOURCE()(CODE OR CODES) OR STATEMENT? OR ASSERTION?
               OR DECLARATION? OR INSTRUCTION?)
S7         27   S3 AND (PROCESSOR? OR MICROPROCESSOR? OR COMPUTER? OR MICR-
              OCOMPUTER? OR (CENTRAL()PROCESSING OR CONTROL OR MICRO?)()(PR-
              OCESSOR? OR COMPUTER? OR UNIT? OR DEVICE?) OR DATA()COLLECTIO-
              N()DEVICE?)
S8          0   S3 AND SPOOF(2N)ASSEMBLER?
File   2:INSPEC 1969-2004/Jan W3
         (c) 2004 Institution of Electrical Engineers
File   6:NTIS 1964-2004/Jan W4
         (c) 2004 NTIS, Intl Cpyrght All Rights Res
File   8:Ei Compendex(R) 1970-2004/Jan W3
         (c) 2004 Elsevier Eng.  Info. Inc.
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Jan W4
         (c) 2004 Inst for Sci Info
File  35:Dissertation Abs Online 1861-2004/Dec
         (c) 2004 ProQuest Info&Learning
File  65:Inside Conferences 1993-2004/Jan W4
         (c) 2004 BLDSC all rts. reserv.
File  92:IHS Intl.Stds.& Specs. 1999/Nov
         (c) 1999 Information Handling Services
File  94:JICST-EPlus 1985-2004/Jan W3
         (c)2004 Japan Science and Tech Corp(JST)
File  95:TEME-Technology & Management 1989-2004/Jan W2
         (c) 2004 FIZ TECHNIK
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Dec
         (c) 2004 The HW Wilson Co.
File 103:Energy SciTec 1974-2004/Jan B1
         (c) 2004 Contains copyrighted material
File 144:Pascal 1973-2004/Jan W3
         (c) 2004 INIST/CNRS
File 202:Info. Sci. & Tech. Abs. 1966-2004/Jan 20
         (c) 2004 EBSCO Publishing
File 233:Internet & Personal Comp. Abs. 1981-2003/Sep
         (c) 2003 EBSCO Pub.
File 239:Mathsci 1940-2003/Feb
         (c) 2003 American Mathematical Society
File 275:Gale Group Computer DB(TM) 1983-2004/Jan 29
         (c) 2004 The Gale Group
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
         (c) 1998 Inst for Sci Info
File 647:CMP  Computer Fulltext 1988-2004/Jan W3
         (c) 2004 CMP Media, LLC
File 674:Computer News Fulltext 1989-2004/Jan W4
         (c) 2004 IDG Communications
File 696:DIALOG Telecom. Newsletters 1995-2004/Jan 15
         (c) 2004 The Dialog Corp.
```

```
Set     Items   Description
S1       2939   AU=(SIMONS, J? OR SIMONS J?)
S2       2633   S1 NOT PY>2000
S3        882   S2 NOT PY<1994
S4          0   S3 AND (ASSEMBLER? OR LINKER? OR COMPILER? OR LANGUAGE()PR-
              OCESSOR)
S5          1   S3 AND (OBJECT OR MACHINE()(CODE OR CODES) OR ASSEMBLY()LA-
              NGUAGE()SOURCE()CODE OR ASSEMBLY()LANGUAGE()INSTRUCTION)
S6          2   S3 AND (SOURCE()(CODE OR CODES) OR STATEMENT? OR ASSERTION?
               OR DECLARATION? OR INSTRUCTION?)
S7         27   S3 AND (PROCESSOR? OR MICROPROCESSOR? OR COMPUTER? OR MICR-
              OCOMPUTER? OR (CENTRAL()PROCESSING OR CONTROL OR MICRO?)()(PR-
              OCESSOR? OR COMPUTER? OR UNIT? OR DEVICE?) OR DATA()COLLECTIO-
              N()DEVICE?)
S8          0   S3 AND SPOOF(2N)ASSEMBLER?
S9       2260   HITACHI()AMERICA
S10         0   S9 AND SPOOF(2N)ASSEMBLER?
File   2:INSPEC 1969-2004/Jan W3
         (c) 2004 Institution of Electrical Engineers
File   6:NTIS 1964-2004/Jan W4
         (c) 2004 NTIS, Intl Cpyrght All Rights Res
File   8:Ei Compendex(R) 1970-2004/Jan W3
         (c) 2004 Elsevier Eng.  Info. Inc.
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Jan W4
         (c) 2004 Inst for Sci Info
File  35:Dissertation Abs Online 1861-2004/Dec
         (c) 2004 ProQuest Info&Learning
File  65:Inside Conferences 1993-2004/Jan W4
         (c) 2004 BLDSC all rts. reserv.
File  92:IHS Intl.Stds.& Specs. 1999/Nov
         (c) 1999 Information Handling Services
File  94:JICST-EPlus 1985-2004/Jan W3
         (c)2004 Japan Science and Tech Corp(JST)
File  95:TEME-Technology & Management 1989-2004/Jan W2
         (c) 2004 FIZ TECHNIK
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Dec
         (c) 2004 The HW Wilson Co.
File 103:Energy SciTec 1974-2004/Jan B1
         (c) 2004 Contains copyrighted material
File 144:Pascal 1973-2004/Jan W3
         (c) 2004 INIST/CNRS
File 202:Info. Sci. & Tech. Abs. 1966-2004/Jan 20
         (c) 2004 EBSCO Publishing
File 233:Internet & Personal Comp. Abs. 1981-2003/Sep
         (c) 2003 EBSCO Pub.
File 239:Mathsci 1940-2003/Feb
         (c) 2003 American Mathematical Society
File 275:Gale Group Computer DB(TM) 1983-2004/Jan 29
         (c) 2004 The Gale Group
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
         (c) 1998 Inst for Sci Info
File 647:CMP  Computer Fulltext 1988-2004/Jan W3
         (c) 2004 CMP Media, LLC
File 674:Computer News Fulltext 1989-2004/Jan W4
         (c) 2004 IDG Communications
File 696:DIALOG Telecom. Newsletters 1995-2004/Jan 15
         (c) 2004 The Dialog Corp.
```

```
Set     Items   Description
S1    1712888   PROCESSOR? OR MICROPROCESSOR? OR COMPUTER? OR MICROCOMPUTE-
            R? OR (CENTRAL()PROCESSING OR CONTROL OR MICRO?)()(PROCESSOR?
            OR COMPUTER? OR UNIT? OR DEVICE?) OR DATA()COLLECTION()DEVICE?
S2     155137   SOURCE()(CODE OR CODES) OR STATEMENT? OR ASSERTION? OR DEC-
            LARATION? OR INSTRUCTION?
S3    2326101   COPYING OR COPY OR COPIES OR STORE? ? OR STORING OR SAVE OR
             SAVING OR KEEP OR KEEPING OR PRESERV? OR MEMORY OR CACHE OR -
            CACHING OR CACHED OR CACHES OR BACKUP OR BACK()UP
S4        336   (TEMPORARY OR IMPERMANENT OR INTERIM OR PROVISIONAL OR SHO-
            RT()(TERM OR RANGE))()(FILE OR FILES)
S5    9808870   APPLY OR APPLIED OR APPLIES OR USE OR USES OR USING OR UTI-
            LIZ? OR EMPLOY? OR IMPLEMENT? OR EXECUT?
S6      14352   ASSEMBLER? OR LINKER? OR COMPILER? OR LANGUAGE()PROCESSOR
S7    3434422   PRODUCE? OR GENERATE? OR REPRODUCE? OR CREATE? OR DEVELOP?
S8       2445   (OBJECT OR MACHINE)()(CODE OR CODES) OR ASSEMBLY()LANGUAGE-
            ()SOURCE()CODE OR ASSEMBLY()LANGUAGE()INSTRUCTION OR ISA
S9         24   S2 AND S3 AND S4
S10         4   S2 AND S4 AND S5 AND S6
S11       314   S6 AND S7 AND S8
S12         2   S9 AND S11
S13        33   S2 AND S4
S14         3   S13 AND S8
S15        25   S9 OR S10 OR S12 OR S14
S16        23   S15 AND IC=G06F?
S17         2   S15 AND MC=T01-F05A
S18        23   S16 OR S17
File 347:JAPIO Oct 1976-2003/Sep(Updated 040105)
        (c) 2004 JPO & JAPIO
File 350:Derwent WPIX 1963-2004/UD,UM &UP=200407
        (c) 2004  Thomson Derwent
```

18/5/1 (Item 1 from file: 347)
DIALOG(R)File 347:JAPIO
(c) 2004 JPO & JAPIO. All rts. reserv.

07328449 **Image available**
METHOD FOR COMPILING ASSEMBLY LANGUAGE CODE FOR **INSTRUCTION** SET ARCHITECTURE INCLUDING NEW **INSTRUCTION** **USING** CONVENTIONAL **ASSEMBLER**

PUB. NO.: 2002-196937 [JP 2002196937 A]
PUBLISHED: July 12, 2002 (20020712)
INVENTOR(s): SIMONS JOHN
APPLICANT(s): HITACHI LTD
APPL. NO.: 2001-328956 [JP 20011328956]
FILED: October 26, 2001 (20011026)
PRIORITY: 00 747824 [US 2000747824], US (United States of America), December 22, 2000 (20001222)
INTL CLASS: **G06F-009/45**

ABSTRACT

PROBLEM TO BE SOLVED: To provide a simple and effective method for compiling an **instruction** of a new **assembly language instruction** set architecture **using** a previous **assembler** .

SOLUTION: An **assembler** expansion **instruction** set architecture **ISA** is formed from the newest **ISA** to which a new **instruction** is added. The assembly of a **source code** output in hybrid of present and new assembly language **instructions** is attained by preprocessing of the **source code** in order to generate a **temporary file** 46 including a previous **instruction** 40a and data designation 41 to each of new assembly **instructions** having **object code** equivalence of the new **instruction** 40b as a data independent function, Thereafter, the **temporary file** 46 is used for the previous **assembler** 40a in order to generate **object codes** corresponding to each of the previous assembly language **instructions** . Resultantly, an **executable** machine language program for the new **ISA** is generated after linking.



18/5/2 (Item 2 from file: 347)
DIALOG(R)File 347:JAPIO
(c) 2004 JPO & JAPIO. All rts. reserv.

06882167 **Image available**
DEVICE AND METHOD FOR TRANSMITTING/RECEIVING ELECTRONIC MAIL WITH ATTACHED FILE

PUB. NO.: 2001-109675 [JP 2001109675 A]
PUBLISHED: April 20, 2001 (20010420)
INVENTOR(s): KANEDA TOSHITAKA
APPLICANT(s): SHARP CORP
APPL. NO.: 11-284148 [JP 99284148]
FILED: October 05, 1999 (19991005)
INTL CLASS: **G06F-013/00** ; H04L-012/54; H04L-012/58

ABSTRACT

PROBLEM TO BE SOLVED: To prevent a file name from being coded in a simple configuration without increasing file capacity when transmitting electronic mail with a file attached.

SOLUTION: A mail text preparing part 1-1 opens a mail transmitting screen and prepares mail to be transmitted on the basis of inputted information. In the case of attaching a file to the mail to be transmitted, when the file name exceeds eight half em alphanumeric characters or is composed of em characters,, a **temporary file** name preparing part 1-2 displays a **temporary file** name input screen. When a user designates a file name within eight half em alphanumeric characters, that designated file name is

complete the input and output operation. If not, the disc cash control unit 400 reads out the sector from the disc and delivers it to CPU via the data buses 570 and 560. In this case, no renewal of the cash **memory** 500 is made. Then, the block is assigned to the cash **memory** 500 at the data write-in and only the blocks not assigned are written on the disc. This assignment is fixed during the file usage period.

**18/5/17 (Item 1 from file: 350)**
DIALOG(R)File 350:Derwent WPIX
(c) 2004 Thomson Derwent. All rts. reserv.

015692813 **Image available**
WPI Acc No: 2003-755002/200371
XRPX Acc No: N03-604956

Source code **conversion in e.g. microcomputer, involves applying** instructions **of** temporary file **to** assembler **to** produce object code **corresponding to old** instructions , **and data forming** object code **for new** instructions

Patent Assignee: HITACHI LTD (HITA ); HITACHI AMERICA LTD (HITA )
Inventor: SIMONS J
Number of Countries: 002 Number of Patents: 002
Patent Family:

| Patent No | Kind | Date | Applicat No | Kind | Date | Week |
|---|---|---|---|---|---|---|
| US 20020083421 | A1 | 20020627 | US 2000747824 | A | 20001222 | 200371 B |
| JP 2002196937 | A | 20020712 | JP 2001328956 | A | 20011026 | 200371 |

Priority Applications (No Type Date): US 2000747824 A 20001222
Patent Details:

| Patent No | Kind | Lan | Pg | Main IPC | Filing Notes |
|---|---|---|---|---|---|
| US 20020083421 | A1 | | 8 | G06F-009/45 | |
| JP 2002196937 | A | | 7 | G06F-009/45 | |

Abstract (Basic): US 20020083421 A1

NOVELTY - The method involves **copying** several **instructions** to a **temporary file** (46). The **instructions** of the **temporary file** are **applied** to an **assembler** to **produce** an **object code** corresponding to the old **instructions** (40a,40b), and a data forming **object code** for new **instructions** .

DETAILED DESCRIPTION - An INDEPENDENT CLAIM is also included for a **source code** assembling method.

**USE** - For converting **source code** to **object code** in microcomputers and microprocessors.

ADVANTAGE - Enables assembling code for new **instruction** -set architecture ( **ISA** ) **using** older **assembler** such that **development** of extended **ISA** can continue concomitant with the **development** of new **assembler** .

DESCRIPTION OF DRAWING(S) - The figure shows a schematic view explaining **source code** conversion.

source file (40)
old assembly **instructions** (40a,40b)
preprocessor (42)
temporary **source code** file (46)
object file (50)
new processor (58)
pp; 8 DwgNo 3/4

Title Terms: SOURCE; CODE; CONVERT; MICROCOMPUTER; **APPLY** ; **INSTRUCTION** ; TEMPORARY; FILE; ASSEMBLE; **PRODUCE** ; OBJECT; CODE; CORRESPOND; **INSTRUCTION** ; DATA; FORMING; OBJECT; CODE; NEW; **INSTRUCTION**
Derwent Class: T01
International Patent Class (Main): **G06F-009/45**
File Segment: EPI

**18/5/18 (Item 2 from file: 350)**
DIALOG(R)File 350:Derwent WPIX
(c) 2004 Thomson Derwent. All rts. reserv.

Priority Applications (No Type Date): US 96625014 A 19960329
Patent Details:
Patent No  Kind Lan Pg  Main IPC  Filing Notes
US 5890165  A  20 G06F-017/30

Abstract (Basic): US 5890165 A
NOVELTY - The output descriptor file (06) **stores** the smallest set of databases which is backed up as a unit without compromising data integrity based on the output of a grouper program. A work item generator (52) performs **back up** schedule for execution of a storage management server computer (50) to search database and **back up** without compromising data integrity.
DETAILED DESCRIPTION - The system has a database client computer (10) connected to the storage management server with the work item generator. The file allocation information of the client is transferred into a **temporary file**, by a set of **instruction** of scripts and executable programs **stored** in the server. The **temporary file** is analyzed by the grouper program. An INDEPENDENT CLAIM is included for automatic databases searching method.
USE - For manufacturing plant and vending management using computer network.
ADVANTAGE - By setting **back up** of files in orderly manner, regular back ups can be obtained without shutting down other system and processes.
DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of database security system.
Descriptor file (06)
Client (10)
Server (50)
Work item generator (52)
pp; 20 DwgNo 1a/10
Title Terms: AUTOMATIC; DATABASE; SEARCH; SYSTEM; MANUFACTURE; PLANT; MANAGEMENT
Derwent Class: T01
International Patent Class (Main): **G06F-017/30**
File Segment: EPI

**18/5/22 (Item 6 from file: 350)**
DIALOG(R)File 350:Derwent WPIX
(c) 2004 Thomson Derwent. All rts. reserv.

010691115 **Image available**
WPI Acc No: 1996-188071/199619
XRPX Acc No: N96-157366
**Database management system capability extending to object oriented languages - compiling object oriented program with pre-** compiler **from database management system into** temporary file **, then compiling** temporary file **with object oriented pre-** compiler **into file acceptable to object oriented** compiler
Patent Assignee: TEXAS INSTR INC (TEXI )
Inventor: ALASHQUR A M
Number of Countries: 001 Number of Patents: 001
Patent Family:
Patent No  Kind  Date  Applicat No  Kind  Date  Week
US 5504885  A  19960402  US 9384841  A  19930629  199619  B

Priority Applications (No Type Date): US 9384841 A 19930629
Patent Details:
Patent No  Kind Lan Pg  Main IPC  Filing Notes
US 5504885  A  12 G06F-017/30

Abstract (Basic): US 5504885 A
The method involves embedding standard query language (SQL) **statements** in an object-oriented program. The object-oriented program with a pre- **compiler** from the database management system compiling into **temporary file** . The **temporary file** is then compiled with

an object-oriented pre- **compiler** into a file acceptable to an object-oriented **compiler** . The file are further compiled with the object-oriented **compiler** into an **executable** file.

The object-oriented program **utilizes** C++. The database management system is a relational database management system which **uses** Oracle.

**USE** /ADVANTAGE - In field of object oriented programming with relational databases. Allows connecting object oriented application programs written in object oriented languages with relational data base. Capable of accessing data **stored** in autonomous, heterogeneous and possibly distributed database management system.

Dwg.2/4

Title Terms: DATABASE; MANAGEMENT; SYSTEM; CAPABLE; EXTEND; OBJECT; ORIENT; LANGUAGE; COMPILE; OBJECT; ORIENT; PROGRAM; PRE; COMPILE; DATABASE; MANAGEMENT; SYSTEM; TEMPORARY; FILE; COMPILE; TEMPORARY; FILE; OBJECT; ORIENT; PRE; COMPILE; FILE; ACCEPT; OBJECT; ORIENT; COMPILE
Derwent Class: T01
International Patent Class (Main): **G06F-017/30**
File Segment: EPI

**18/5/23 (Item 7 from file: 350)**
DIALOG(R)File 350:Derwent WPIX


007261299
WPI Acc No: 1987-258306/198737
XRPX Acc No: N87-193408

**On-line presentation** implementation **for information processing - has** compiler **which** creates **file identifying image data to be used in presentation**

Patent Assignee: INT BUSINESS MACHINES CORP (IBMC ); IBM CORP (IBMC )
Inventor: GAITHER W D; GIOVANNETTI L T; GRAFE R J; HALL L F; MEYER G P; PANCOAST S T; GIOVANNETT L T
Number of Countries: 006 Number of Patents: 005
Patent Family:

| Patent No | Kind | Date | Applicat No | Kind | Date | Week | |
|---|---|---|---|---|---|---|---|
| EP 237014 | A | 19870916 | EP 87103377 | A | 19870310 | 198737 | B |
| US 4864516 | A | 19890905 | US 86837996 | A | 19860310 | 198945 | |
| CA 1271564 | A | 19900710 | | | | 199033 | |
| EP 237014 | B1 | 19940713 | EP 87103377 | A | 19870310 | 199427 | |
| DE 3750188 | G | 19940818 | DE 3750188 | A | 19870310 | 199432 | |
| | | | EP 87103377 | A | 19870310 | | |

Priority Applications (No Type Date): US 86837996 A 19860310
Cited Patents: 1.Jnl.Ref; A3...9026; No-SR.Pub
Patent Details:

| Patent No | Kind | Lan | Pg | Main IPC | Filing Notes |
|---|---|---|---|---|---|
| EP 237014 | A | E | 14 | | |
| Designated States (Regional): DE FR GB IT | | | | | |
| US 4864516 | A | | 11 | | |
| EP 237014 | B1 | E | 14 | G06F-015/72 | |
| Designated States (Regional): DE FR GB IT | | | | | |
| DE 3750188 | G | | | G06F-015/72 | Based on patent EP 237014 |

Abstract (Basic): EP 237014 A

The data assembly method comprises entering into the system image data **using** the data input device. A number of control commands are entered for displaying the image data separately from the image data. A pictorial representation of the image data is displayed **using** the control commands. The selected control commands are compiled. **Object code** files are **created** from the compiled commands. A **temporary file** is **created** identifying the image data required in the on-line presentation. A compressed image data file is **created** .

Each APA image **created** is identified and **stored** . Each ASCII image **created** is identified and **stored** .

1/7

Title Terms: LINE; PRESENT; **IMPLEMENT** ; INFORMATION; PROCESS; COMPILE;

```
File 411:DIALINDEX(R)

DIALINDEX(R)
   (c) 2004 The Dialog Corporation plc

*** DIALINDEX search results display in an abbreviated ***
*** format unless you enter the SET DETAIL ON command. ***
?sf all
   You have 556 files in your file list.
   (To see banners, use SHOW FILES command)
?s spoof()assembler

Your SELECT statement is:
   s spoof()assembler

          Items   File
          -----   ----
       Examined  50 files
       Examined 100 files
       Examined 150 files
       Examined 200 files
       Examined 250 files
       Examined 300 files
       Examined 350 files
       Examined 400 files
       Examined 450 files
       Examined 500 files
       Examined 550 files

   No files have one or more items; file list includes 556 files.
```

```
Set     Items   Description
S1          0   SPOOF()ASSEMBLER?
S2       2939   AU=(SIMONS, J? OR SIMONS J?)
S3       2633   S2 NOT PY>2000
S4       2632   S3 NOT PD>20001222
S5          0   S4 AND SPOOF?()ASSEMBLER?
S6          0   S4 AND ASSEMBLER?
S7          0   S4 AND COMPUTER()LANGUAGE
S8        882   S4 NOT PY<1994
S9          0   S8 AND ASSEMBLY()LANGUAGE
S10         0   S8 AND COMPUTER()LANGUAGE?
File   2:INSPEC 1969-2004/Jan W3
         (c) 2004 Institution of Electrical Engineers
File   6:NTIS 1964-2004/Jan W4
         (c) 2004 NTIS, Intl Cpyrght All Rights Res
File   8:Ei Compendex(R) 1970-2004/Jan W3
         (c) 2004 Elsevier Eng.  Info. Inc.
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Jan W4
         (c) 2004 Inst for Sci Info
File  35:Dissertation Abs Online 1861-2004/Dec
         (c) 2004 ProQuest Info&Learning
File  65:Inside Conferences 1993-2004/Jan W4
         (c) 2004 BLDSC all rts. reserv.
File  92:IHS Intl.Stds.& Specs. 1999/Nov
         (c) 1999 Information Handling Services
File  94:JICST-EPlus 1985-2004/Jan W3
         (c)2004 Japan Science and Tech Corp(JST)
File  95:TEME-Technology & Management 1989-2004/Jan W2
         (c) 2004 FIZ TECHNIK
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Dec
         (c) 2004 The HW Wilson Co.
File 103:Energy SciTec 1974-2004/Jan B1
         (c) 2004 Contains copyrighted material
File 144:Pascal 1973-2004/Jan W3
         (c) 2004 INIST/CNRS
File 202:Info. Sci. & Tech. Abs. 1966-2004/Jan 20
         (c) 2004 EBSCO Publishing
File 233:Internet & Personal Comp. Abs. 1981-2003/Sep
         (c) 2003 EBSCO Pub.
File 239:Mathsci 1940-2003/Feb
         (c) 2003 American Mathematical Society
File 275:Gale Group Computer DB(TM) 1983-2004/Jan 29
         (c) 2004 The Gale Group
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
         (c) 1998 Inst for Sci Info
File 647:CMP  Computer Fulltext 1988-2004/Jan W3
         (c) 2004 CMP Media, LLC
File 674:Computer News Fulltext 1989-2004/Jan W4
         (c) 2004 IDG Communications
File 696:DIALOG Telecom. Newsletters 1995-2004/Jan 15
         (c) 2004 The Dialog Corp.
```

```
Set     Items   Description
S1    3239956   PROCESSOR? OR MICROPROCESSOR? OR COMPUTER? OR MICROCOMPUTE-
            R? OR (CENTRAL()PROCESSING OR CONTROL OR MICRO?)()(PROCESSOR?
            OR COMPUTER? OR UNIT? OR DEVICE?) OR DATA()COLLECTION()DEVICE?
S2     307864   SOURCE()(CODE OR CODES) OR STATEMENT? OR ASSERTION? OR DEC-
            LARATION? OR INSTRUCTION?
S3    1342515   COPYING OR COPY OR COPIES OR STORE? ? OR STORING OR SAVE OR
             SAVING OR KEEP OR KEEPING OR PRESERV? OR MEMORY OR CACHE OR -
            CACHING OR CACHED OR CACHES OR BACKUP OR BACK()UP
S4        134   (TEMPORARY OR IMPERMANENT OR INTERIM OR PROVISIONAL OR SHO-
            RT()(TERM OR RANGE))()(FILE OR FILES)
S5    9539933   APPLY OR APPLIED OR APPLIES OR USE OR USES OR USING OR UTI-
            LIZ? OR EMPLOY? OR IMPLEMENT? OR EXECUT?
S6      65146   ASSEMBLER? OR LINKER? OR COMPILER? OR LANGUAGE()PROCESSOR
S7    6334112   PRODUCE? OR GENERATE? OR REPRODUCE? OR CREATE? OR DEVELOP?
S8      23497   (OBJECT OR MACHINE)()(CODE OR CODES) OR ASSEMBLY()LANGUAGE-
            ()SOURCE()CODE OR ASSEMBLY()LANGUAGE()INSTRUCTION OR ISA
S9          1   S2 AND S3 AND S4
S10         0   S2 AND S4 AND S5 AND S6
S11         0   S2 AND S4 AND S6
S12      1019   S6 AND S7 AND S8
S13       719   S1 AND S12
S14         0   S13 AND S2 AND S4
S15       285   S13 AND S2
S16         0   S13 AND S4
S17         0   S15 AND S4
S18       230   S15 AND S5
S19       230   S18 AND S6
S20        67   S19 AND S3
S21         0   S19 AND S4
S22        68   S9 OR S20
S23        64   S22 NOT PY>2000
S24        64   S23 NOT PD>20001222
S25        59   RD (unique items)
File   8:Ei Compendex(R) 1970-2004/Jan W3
         (c) 2004 Elsevier Eng.  Info. Inc.
File  35:Dissertation Abs Online 1861-2004/Dec
         (c) 2004 ProQuest Info&Learning
File 202:Info. Sci. & Tech. Abs. 1966-2004/Jan 20
         (c) 2004 EBSCO Publishing
File  65:Inside Conferences 1993-2004/Jan W4
         (c) 2004 BLDSC all rts. reserv.
File   2:INSPEC 1969-2004/Jan W3
         (c) 2004 Institution of Electrical Engineers
File 233:Internet & Personal Comp. Abs. 1981-2003/Sep
         (c) 2003 EBSCO Pub.
File  94:JICST-EPlus 1985-2004/Jan W3
         (c)2004 Japan Science and Tech Corp(JST)
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Dec
         (c) 2004 The HW Wilson Co.
File  95:TEME-Technology & Management 1989-2004/Jan W2
         (c) 2004 FIZ TECHNIK
File 583:Gale Group Globalbase(TM) 1986-2002/Dec 13
         (c) 2002 The Gale Group
```

**25/5/1 (Item 1 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


05444452 E.I. No: EIP99124950381
**Title:** Compiler **-driven** cached **code compression schemes for embedded ILP** processors
Author: Larin, Sergei Y.; Conte, Thomas M.
Corporate Source: North Carolina State Univ, Raleigh, NC, USA
Conference Title: Proceedings of the 1999 32nd Annual ACM/IEEE International Symposium on Microarchitecture, MICRO-32
Conference Location: Haifa, Isr Conference Date: 19991116-19991118
Sponsor: IEEE TC-MARCH; ACM SIGMICRO
E.I. Conference No.: 56105
Source: Proceedings of the Annual International Symposium on Microarchitecture 1999. p 82-92
Publication Year: 1999
CODEN: PSMIE7 ISSN: 1072-4451
Language: English
Document Type: JA; (Journal Article) Treatment: T; (Theoretical)
Journal Announcement: 0002W1
Abstract: During the last 15 years, embedded systems have grown in complexity and performance to rival desktop systems. The architectures of these systems present unique challenges to **processor** microarchitecture, including **instruction** encoding and **instruction** fetch processes. This paper presents new techniques for reducing embedded system code size without reducing functionality. This approach is to extract the pipeline decoder logic for an embedded VLIW **processor** in software at system **development** time. The code size reduction is achieved by Huffman compressing or tailor encoding the **ISA** of the original program. Some interesting results were found. In particular, the degree of compression for the ROM doesn't translate into an improvement in **instructions** delivered per cycle. Experiments found that when the misspredict ion penalty of the added Huffman decoder stage was taken into account, a Tailored **ISA** approach **produced** higher performance. Methods that compress the entire operation **using** Huffman encodings, and decompress at ICache hit time still achieved a median performance advantage, while providing higher ROM size savings. All results were **generated** by an optimizing **compiler** and tool suite, and presented for an encoding similar to the Intel/HP IA-64 architecture. (Author abstract) 27 Refs.
Descriptors: **Microprocessor** chips; **Computer** architecture; Integrated circuit layout; Program **compilers** ; Buffer storage; Data compression; Embedded systems; Pipeline processing systems; Optimization
Identifiers: **Cache** code compression methods
Classification Codes:
714.2 (Semiconductor Devices & Integrated Circuits); 723.1 (Computer Programming); 722.1 (Data Storage, Equipment & Techniques); 722.4 (Digital Computers & Systems); 921.5 (Optimization Techniques)
714 (Electronic Components); 722 (Computer Hardware); 723 (Computer Software); 921 (Applied Mathematics)
71 (ELECTRONICS & COMMUNICATIONS); 72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

**25/5/2 (Item 2 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


05371442 E.I. No: EIP99094776315
**Title: Compiling Esterel into sequential code**
Author: Edwards, Stephen A.
Corporate Source: Synopsys, Inc, Mountain View, CA, USA
Conference Title: Proceedings of the 1999 7th International Conference on Hardware/Software Codesign (CODES'99)
Conference Location: Rome, Italy Conference Date: 19990503-19990505
Sponsor: ACM SIGDA; IEEE Computer Society; ACM SIGSOFT; IFIP WG 10.5

E.I. Conference No.: 55455
Source: Hardware/Software Codesign - Proceedings of the International Workshop 1999. p 147-151
Publication Year: 1999
CODEN: 85PQAI
Language: English
Document Type: JA; (Journal Article) Treatment: T; (Theoretical)
Journal Announcement: 9911W1
Abstract: This paper presents a novel **compiler** for Esterel, a concurrent synchronous imperative language. It **generates** fast, small **object code** by compiling away concurrency, producing a single C function requiring no operating system support for threads. It translates an Esterel program into an acyclic concurrent control-flow graph from which code is synthesized that runs **instructions** in an order respecting inter-thread communication. Exceptions and preemption constructs become conditional branches. Variables **save** control state; conditional branches restore it. Although designed for Esterel, this approach could be **applied** to compiling other synchronous concurrent languages. (Author abstract) 8 Refs.
Descriptors: **Computer** architecture; **Computer** programming languages; Program **compilers** ; Codes (symbols); **Computer** operating systems; C (programming language); Program translators
Identifiers: Esterel programming language; Sequential codes
Classification Codes:
723.1.1 (Computer Programming Languages)
723.1 (Computer Programming); 723.2 (Data Processing)
722 (Computer Hardware); 723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

**25/5/3 (Item 3 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


05365360 E.I. No: EIP99094789940
**Title: New mediaprocessor and its image computing library**
Author: Kim, Donglok; Stotland, Inga; Kim, Yongmin
Corporate Source: Univ of Washington, Seattle, WA, USA
Conference Title: Proceedings of the 1999 Medical Imaging - Image Dispaly
Conference Location: San Diego, CA, USA Conference Date: 19990221-19990223
Sponsor: SPIE
E.I. Conference No.: 55282
Source: Proceedings of SPIE - The International Society for Optical Engineering v 3658 1999. p 220-229
Publication Year: 1999
CODEN: PSISDG ISSN: 0277-786X
Language: English
Document Type: JA; (Journal Article) Treatment: A; (Applications); G; (General Review)
Journal Announcement: 9910W5
Abstract: High-performance mediaprocessors that aim at high-level language programming without sacrificing much would be very desirable in many applications including medical imaging. Media Accelerated **Processor** (MAP) 1000 that is jointly being **developed** by Hitachi, Ltd. and Equator Technologies, Inc. is one of such next-generation mediaprocessors. We present the two main issues in programming these mediaprocessors, i.e., **using** C intrinsics (hinting to the **compiler** a desired **assembly language instruction** in a C program so that the **compiler** tries to **use** the hinted **instruction** in the compiled code) and data flow control, which still requires a high degree of expertise in the detailed architectural features including many low-level **instructions** , handling input/output data transfers, and in-depth understanding of the algorithm. To ease the programming burden and allow flexible and efficient deployment of the MAP-based target systems, we have **developed** the MAP University of Washington Image Computing Library (UWICL) for the MAP1000. Currently, it consists of 105 functions. The UWICL functions effectively decouple the data flow control and data processing in a flexible two-layered software

structure, where the upper layer is responsible for the data transfer between on-chip **cache** and off-chip **memory** by **utilizing** the on-chip DMA controller in a double-buffering scheme and the lower layer performs the data processing. This hierarchy allows the flexibility to **use** the UWICL modules depending on how the application is **implemented** and/or the level of user's experience in programming the MAP1000 mediaprocessor. (Author abstract) 22 Refs.

Descriptors: Multimedia systems; C (programming language); Data flow analysis; High level languages; Data transfer; Algorithms; Image processing ; Data processing; Medical imaging; **Computer** software

Identifiers: Mediaprocessor; Direct **memory** access controller; Image computing library

Classification Codes:

723.1.1 (Computer Programming Languages)

723.5 (Computer Applications); 723.1 (Computer Programming); 723.2 (Data Processing)

723 (Computer Software)

72 (COMPUTERS & DATA PROCESSING)

**25/5/4 (Item 4 from file: 8)**



05254728 E.I. No: EIP99034617898

**Title: Computational infrastructure of analogic CNN computing - Part I: The CNN-UM chip prototyping system**

Author: Roska, Tamas; Zarandy, Akos; Zold, Sandor; Foldesy, Peter; Szolgay, Peter

Corporate Source: Hungarian Acad of Sciences, Budapest, Hung

Source: IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications v 46 n 2 Feb 1999. p 261-268

Publication Year: 1999

CODEN: ITCAEX ISSN: 1057-7122

Language: English

Document Type: JA; (Journal Article) Treatment: A; (Applications); T; (Theoretical)

Journal Announcement: 9905W3

Abstract: A new computational paradigm is emerging for spatio-temporal problems: analogic cellular neural/nonlinear network (CNN) array-computing. It has all the software and hardware ingredients of the **stored** programmable **computer** (high-level language, **compiler** , operating system, assembly and **machine code** , analogic **central processing unit** (CPU), analog and digital **memory** , peripherals, etc.). The elementary **instructions** and programming techniques, however, are drastically different from any other **computers** . These elementary **instructions** represent complex spatio-temporal nonlinear dynamic phenomena, including all the standard and exotic properties of simple image processing operators, as well as waves, patterns, and evolving systems. Meanwhile, a computational infrastructure has also emerged left bracket 6 right bracket interfacing this revolutionary computing technology to digital systems as well as enabling the **development** of high-level software. The main goal was to provide easy-to- **use** tools and hardware interfacing elements and putting most of the sophistication in the chips and chip sets. (Author abstract) 17 Refs.

Descriptors: Cellular neural networks; Algorithms; **Computer** systems programming; Interfaces ( **computer** ); **Computer** architecture; **Microprocessor** chips; Integrated circuit layout

Identifiers: Computational infrastructure; Analogic algorithms

Classification Codes:

723.4 (Artificial Intelligence); 723.1 (Computer Programming); 722.2 (Computer Peripheral Equipment)

723 (Computer Software); 921 (Applied Mathematics); 722 (Computer Hardware)

72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

**25/5/5 (Item 5 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


04426184 E.I. No: EIP96063214949
**Title: VCODE: a retargetable, extensible, very fast dynamic code generation system**
Author: Engler, Dawson R.
Corporate Source: Massachusetts Inst of Technology, Cambridge, MA, USA
Conference Title: Proceedings of the ACM SIGPLAN'96 Conference on Programming Language Design and Implementation, PLDI
Conference Location: Philadelphia, PA, USA Conference Date: 19960521-19960524
Sponsor: SIGPLAN
E.I. Conference No.: 44780
Source: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) 1996. ACM, New York, NY, USA. p 160-170
Publication Year: 1996
CODEN: PSPIEM
Language: English
Document Type: CA; (Conference Article) Treatment: A; (Applications)
Journal Announcement: 9608W2
Abstract: Dynamic code generation is the creation of **executable** code at runtime. Such 'on-the-fly' code generation is a powerful technique, enabling applications to **use** runtime information to improve performance by up to an order of magnitude. Unfortunately, previous general-purpose dynamic code generation systems have been either inefficient or non-portable. We present VCODE, a retargetable, extensible, very fast dynamic code generation system. An important feature of VCODE is that it **generates machine code** 'in-place' without the **use** of intermediate data structures. Eliminating the need to construct and consume an intermediate representation at runtime makes VCODE both efficient and extensible. VCODE dynamically **generates** code at an approximate cost of six to ten **instructions** per **generated instruction** , making it over an order of magnitude faster than the most efficient general-purpose code generation system in the literature. Dynamic code generation is relatively well known within the **compiler** community. However, due in large part to the lack of a publicly available dynamic code generation system, it has remained a curiosity rather than a widely used technique. A practical contribution of this work is the free, unrestricted distribution of the VCODE system, which currently runs on the MIPS, SPARC, and Alpha architectures. (Author abstract) 24 Refs.
Descriptors: Program **compilers** ; Codes (symbols); Program interpreters; Reduced **instruction** set computing; **Computer** architecture; **Computer** hardware; Optimization; Boundary conditions; Program **assemblers** ; **Computer** programming languages
Identifiers: Dynamic code generation system; **Executable** code; Runtime information; Binary **instruction** ; **Cache** coherence
Classification Codes:
723.1.1 (Computer Programming Languages)
723.1 (Computer Programming); 723.2 (Data Processing); 921.5 (Optimization Techniques)
723 (Computer Software); 722 (Computer Hardware); 921 (Applied Mathematics)
72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

**25/5/6 (Item 6 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


04265679 E.I. No: EIP95102889589
**Title: Highlights from nhc - a space-efficient Haskell** compiler
Author: Rojemo, Niklas
Corporate Source: Chalmers Univ of Technology, Goteborg, Sweden
Conference Title: Conference Record of Conference on Functional Programming Languages and Computer Architecture

Conference Location: La Jolla, CA, USA Conference Date: 19950625-19950628
Sponsor: ACM SIGPLAN; ACM SIGARCH; IFIP
E.I. Conference No.: 43744
Source: Conf Rec Conf Funct Program Lang Comput Archit 1995. ACM. p 282-292
Publication Year: 1995
Language: English
Document Type: CA; (Conference Article) Treatment: A; (Applications); T ; (Theoretical)
Journal Announcement: 9512W1
Abstract: Self-compiling **implementations** of Haskell, i.e., those written in Haskell, have been and, except one, are still space consuming monsters. **Object code** size for the **compilers** themselves are 3-8 MByte, and they need 12-20 MByte to recompile themselves. One reason for the huge demands for **memory** is that the main goal for these **compilers** is to **produce** fast code. However, the **compiler** described in this paper, called nhc for Nearly a Haskell **Compiler** , is the above mentioned exception. This **compiler** concentrates on **keeping memory** usage down, even at a cost in time. The code **produce** is not fast but nhc is usable, and the resulting programs can be run, on **computers** with small **memory** . This paper describes some of the **implementation** choices done, in the Haskell part of the **source code** , to reduce **memory** consumption in nhc. It is possible to **use** these also in other Haskell **compilers** with no, or very small, changes to their run-time systems. Time is neither the main focus of nhc nor of this paper, but there is nevertheless a small section about the speed of nhc. The most notable observation concerning speed is that nhc spends approximately half the time processing interface files, which is much more than needed in the type checker. Processing interface files is also the most space consuming part of nhc in most cases. It is only when compiling source files with large sets of mutually recursive functions that more **memory** is needed to type check than to process interface files. (Author abstract) 16 Refs.
Descriptors: Program **compilers** ; Data storage equipment; Codes (symbols) ; Interfaces ( **computer** ); File organization; Recursive functions; Optimization
Identifiers: Glasgow Haskell **compiler** ; Runtime systems; **Memory** usage; Interface files
Classification Codes:
723.1 (Computer Programming); 722.1 (Data Storage, Equipment & Techniques); 723.2 (Data Processing); 722.2 (Computer Peripheral Equipment); 721.1 (Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory); 921.5 (Optimization Techniques)
723 (Computer Software); 722 (Computer Hardware); 721 (Computer Circuits & Logic Elements); 921 (Applied Mathematics)
72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

**25/5/7 (Item 7 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


03405439 E.I. Monthly No: EI9204044571
**Title: Dual-ALU CRISC architecture and its compiling technique.**
Author: Chou, Hong-Chich; Chung, Chung-Ping; Cheng, Shyi-Chyi
Corporate Source: Natl Chiao Tung Univ, Hsinchu, Taiwan
Source: Computers & Electrical Engineering v 17 n 4 1991 p 297-312
Publication Year: 1991
CODEN: CPEEBQ ISSN: 0045-7906
Language: English
Document Type: JA; (Journal Article) Treatment: T; (Theoretical); A; (Applications)
Journal Announcement: 9204
Abstract: As semiconductor technology advances, more devices can be accommodated in a single VLSI chip. The feasibility of putting multifunctional units in a chip is then worth studying. Such an approach,

however, will face software and hardware difficulties (and also tradeoffs). CRISC is a 32-bit single-chip VLSI **processor** architecture achieving high performance by means of RISC and multiple functional unit approaches. Dual-ALUs are used to **execute** **instructions** concurrently for fine-grained parallelism. Up to three **instructions** can be **executed** simultaneously by CRISC. Here, CRISC architecture design considerations and **instruction** **cache** scheme are investigated. Final microarchitecture and its incorporated software technique to **produce** **object** **code** for fine-grained parallel **execution** are described; its upper bound performance is estimated by an architectural model. A preliminary evaluation of the CRISC is also conducted, showing most satisfying results. (Author abstract) 30 Refs.

Descriptors: **COMPUTER** ARCHITECTURE--*Reduced **Instruction** Set Computing; INTEGRATED CIRCUITS, VLSI; **COMPUTER** OPERATING SYSTEMS--Program **Compilers** ; **COMPUTER** PROGRAMMING--Algorithms

Identifiers: VLSI CHIPS; CRISC ARCHITECTURE

Classification Codes:

722 (Computer Hardware); 723 (Computer Software); 713 (Electronic Circuits); 714 (Electronic Components)

72 (COMPUTERS & DATA PROCESSING); 71 (ELECTRONICS & COMMUNICATIONS)

**25/5/8 (Item 8 from file: 8)**

DIALOG(R)File 8:Ei Compendex(R)



03057747 E.I. Monthly No: EI9105051120

**Title:** Using **rewriting** **techniques** **to** produce **code** **generators** **and** **proving** **them** **correct.**

Author: Despland, Annie; Mazaud, Monique; Rakotozafy, Raymond

Corporate Source: Univ d'Orleans, Orleans, Fr

Source: Science of Computer Programming v 15 n 1 Nov 1990 p 15-54

Publication Year: 1990

CODEN: SCPGD4 ISSN: 0167-6423

Language: English

Document Type: JA; (Journal Article) Treatment: T; (Theoretical)

Journal Announcement: 9105

Abstract: A major problem in deriving a **compiler** from a formal definition is the production of correct and efficient **object** **code** . In this context, we propose a solution to the problem of code-generator generation. Our approach is based on a target machine description where the basic concept used (storage classes, access modes, access classes and **instructions** ) are hierarchically described by tree patterns. These tree patterns are terms of an abstract data type. The program intermediate representation (input to the code generator) is a term of the same abstract data type. The code generation process is based on access modes and **instruction** template-driven rewritings. The result is that each program **instruction** is reduced to a sequence of elementary machine **instructions** , each of them representing an instance of an **instruction** template. The axioms of the abstract data type are used to prove that the rewritings **preserve** the semantics of the intermediate representation. (Author abstract) 21 Refs.

Descriptors: **COMPUTER** OPERATING SYSTEMS--*Program **Compilers** ; **COMPUTER** PROGRAMMING LANGUAGES

Identifiers: CODE-GENERATOR GENERATION; **COMPILER** GENERATOR; TEMPLATE-DRIVEN REWRITINGS; TARGET MACHINE DESCRIPTION

Classification Codes:

723 (Computer Software)

72 (COMPUTERS & DATA PROCESSING)

**25/5/9 (Item 9 from file: 8)**

DIALOG(R)File 8:Ei Compendex(R)



02732349 E.I. Monthly No: EI8904031763

**Title: Multiprocessor software for the CYBERPLUS high performance system.**

Author: Katz, Shlomo; Ray, Wayne A.; Walder, Gabriel
Corporate Source: Control Data Corp, Bloomington, MN, USA
Conference Title: Proceedings of the International Conference on Vector and Parallel Processors in Computational Science III
Conference Location: Liverpool, Engl Conference Date: 19870825
Sponsor: US Army European Research Office, USA
E.I. Conference No.: 11993
Source: Parallel Computing v 8 n 1-3 Oct 1988. p 231-244
Publication Year: 1988
CODEN: PACOEJ ISSN: 0167-8191
Language: English
Document Type: JA; (Journal Article) Treatment: A; (Applications)
Journal Announcement: 8904
Abstract: This paper describes the results of the research for **implementing** applications for concurrent **execution** on the CYBERPLUS multiparallel **computer** system. Three layers of parallelism are built into this system: **instruction** , computation and functional levels. The paper contains a short summary of the hardware, and the different layers of the software for multiprocessor systems. They are based on the previously **developed** CPFTN (Cyberplus Fortran **Compiler** ) **compiler** , which **produces** optimized **object code** for a single CYBERPLUS **processor** . The input to CPFTN consists of an entire kernel containing all subroutines and functions loaded into a single **processor** . An interprocessor communications subsystem provides the low-level component of the multiprocessor system and contains the data transfer and synchronization capabilities. The high-level component consists of extensions to FORTRAN (CPMFTN, Cyberplus Multiprocessor Fortran) for characterizing the data shared between **processors** and **implementing** concepts specially adapted to the private **memory** architecture of the CYBERPLUS as well as to the specific needs of the user community targeted by this product. The resulting scheme is a large grain size, demand driven, data flow type. The first version of CPMFTN under **development** is described followed by several simple application paradigms and discussion of the merits of possible future extensions. (Edited author abstract) 14 Refs.
Descriptors: **COMPUTER** OPERATING SYSTEMS--*Program **Compilers** ; **COMPUTER** SYSTEMS, DIGITAL--Parallel Processing; **COMPUTER** PROGRAMMING LANGUAGES--FORTRAN; **COMPUTER** SOFTWARE
Identifiers: MULTIPROCESSOR SOFTWARE; CYBERPLUS MULTIPARALLEL **COMPUTER** SYSTEM; CPFTN **COMPILER** ; DATA FLOW COMPUTING; CYBERPLUS MULTIPROCESSOR FORTRAN; CPMFTN **COMPUTER** LANGUAGE
Classification Codes:
723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

**25/5/10 (Item 10 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


02105055 E.I. Monthly No: EIM8607-047505
**Title: CODE OPTIMIZATION IN** COMPILER **DESIGN.**
Author: Hubbart, D. Barbara; Schroeder, Charles
Corporate Source: Louisiana State Univ-Shreveport, Shreveport, LA, USA
Conference Title: 1983 ACM Computer Science Conference and SIGCSE Symposium.
Conference Location: Orlando, FL, USA Conference Date: 19830215
Sponsor: ACM, New York, NY, USA
E.I. Conference No.: 03890
Source: Publ by ACM, New York, NY, USA p 84
Publication Year: 1983
ISBN: 0-89791-092-3
Language: English
Document Type: PA; (Conference Paper)
Journal Announcement: 8607
Abstract: Producing an efficient object module was a major objective in the design of one of the first **compilers** ever written. The process that transforms **source code** into efficient **machine code** is known as

**compiler** optimization. Optimizing **compilers** play a major role in any high-level language system and **compiler** writers have a vast number of optimization methods from which to choose. The techniques range from those that are simple to those that are very complex, from techniques that are **applied** to small units of code to techniques that are **applied** to the entire source module. Each method can speed up **execution** time, reduce **memory** requirements, or both. Choice of optimizing techniques often depends on the established goals for the **compiler** , the structure of the language the **compiler** is to be written in, and the machine on which it is to run. An example of an optimizing **compiler** that performs a thorough analysis of the **source code** and **produces** the most efficient **object code** is the FORTRAN IV (H Extended) **compiler** . This **compiler implements** text and branch optimization methods, full register assignment, and does a thorough analysis of the loop structure of the object module. (Author abstract)

Descriptors: **COMPUTER** OPERATING SYSTEMS--*Program **Compilers** ; **COMPUTER** PROGRAMMING LANGUAGES--FORTRAN

Identifiers: **COMPILER** DESIGN; CODE OPTIMIZATION; ABSTRACT ONLY

Classification Codes:
723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

**25/5/11 (Item 11 from file: 8)**

DIALOG(R)File 8:Ei Compendex(R)


01839729 E.I. Monthly No: EI8512114097 E.I. Yearly No: EI85021147

**Title: VALIDITY AND** PRESERVING **REGISTERS.**

Author: Anon

Source: IBM Technical Disclosure Bulletin v 28 n 6 Nov 1985 p 2526-2529

Publication Year: 1985

CODEN: IBMTAA ISSN: 0018-8689

Language: ENGLISH

Document Type: JA; (Journal Article) Treatment: A; (Applications); T; (Theoretical)

Journal Announcement: 8512

Abstract: A method is disclosed for optimizing the length of intermediate text **produced** by an **executing** multiple-pass **compiler** converting source to **object code** . The method comprises the following steps: (a) in a first pass assigning symbolic registers to each canonical expression (opcode and operands) of qualifying **source code instructions** , (b) assuring that each symbolic register holds the most current value of the canonical expression, and (c) replacing a computation with a register-to-register load at selected points in load, **store** , and add sequences. More specifically, an intermediate language, as **produced** , by the front end, **uses** two types of variables: **memory** operands, which correspond to the original variables declared by the user in his source program, and **compiler** temporaries, referred to as symbolic registers. The front end assumes an unlimited number of symbolic registers, and **produces** as many as it needs. Later, in the back end, symbolic registers get transformed into either string temporaries or real (machine) registers.

Descriptors: **COMPUTER** OPERATING SYSTEMS--*Program **Compilers** ; CODES, SYMBOLIC

Identifiers: INTERMEDIATE TEXT; MULTIPLE-PASS **COMPILER** ; SYMBOLIC REGISTERS; **SOURCE CODE** ; **MEMORY** OPERANDS

Classification Codes:
722 (Computer Hardware); 723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

**25/5/12 (Item 12 from file: 8)**

DIALOG(R)File 8:Ei Compendex(R)


01794128 E.I. Monthly No: EI8509075730 E.I. Yearly No: EI85021122

**Title: GENERAL-PURPOSE CROSSASSEMBLER FOR 8-BIT** MICROPROCESSORS .

Author: Newman, Ian A.; Winder, Russel L.
Corporate Source: Loughborough Univ of Technology, Dep of Computer Studies, Loughborough, Engl
Source: Microprocessors and Microsystems v 9 n 5 Jun 1985 p 234-240
Publication Year: 1985
CODEN: MIMID5 ISSN: 0141-9331
Language: ENGLISH
Document Type: JA; (Journal Article) Treatment: T; (Theoretical)
Journal Announcement: 8509
Abstract: A study of **microprocessor instruction** sets is presented. The techniques devised to **implement** a general-purpose crossassembler for 8-bit machines are described. The program has been **implemented** on a VAX-11/750 under Berkeley Unix 4. 1 BSD. Data structures that are used to **store** definitions of the assembly languages and the **machine code instructions** which are to be **generated** are explained. 9 refs.
Descriptors: **COMPUTER** OPERATING SYSTEMS--*Program **Assemblers** ; **COMPUTERS** , **MICROPROCESSOR** ; DATA PROCESSING--Data Structures
Identifiers: GENERAL-PURPOSE CROSSASSEMBLER; BERKELEY UNIX; **INSTRUCTION** SET; **MACHINE CODE INSTRUCTIONS**
Classification Codes:
723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

**25/5/13 (Item 13 from file: 8)**
DIALOG(R)File 8:Ei Compendex(R)


00400553 E.I. Monthly No: EI7410060994
**Title: ALL APPLICATIONS DIGITAL** COMPUTER .
Author: Nissen, Stanley M.; Wallach, Steven J.
Corporate Source: Raytheon Co, Bedford, Mass
Source: Symp on High-Level-Lang Comput Archit, Univ of Md, College Park, Nov 7-8 1973 p 43-51. Available from ACM, New York, 1973
Publication Year: 1973
Language: ENGLISH
Journal Announcement: 7410
Abstract: Described is the all applications digital **computer** data processing element (DPE), built for the Navy to be used for multiplatform (air, land, seas, and underseas) applications in the 1978-1990 time frame. It was the intent of the designers of this **computer** to: achieve a Syntax-oriented internal architecture so that application programs written in a higher level language (HLL) would run efficiently, and to make **use** of a **memory** heiracy system to minimize the number of page faults **generated** by the virtual addressing mechanisms. The HLL language used as a standard to model the internal computational structures was APL. Studies convinced the designers that the most efficient **machine code** and most efficient program **execution** is achieved by the most compact representation of a problem in a HLL. This led to the choice of APL. Thus, the majority of APL operators are part of the **instruction** set of the DPE and the capability exists to directly **execute** operations on vectors and matrices without resorting to **compiler generated** loops.
Descriptors: **COMPUTER** ARCHITECTURE
Classification Codes:
723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

**25/5/14 (Item 1 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


01715192 ORDER NO: AADAA-I0800424
**Generating multithreaded code from parallel Haskell for symmetric multiprocessors**
Author: Caro, Alejandro
Degree: Ph.D.

Year: 1999
Corporate Source/Institution: Massachusetts Institute of Technology ( 0753)
Supervisor: Arvind
Source: VOLUME 60/10-B OF DISSERTATION ABSTRACTS INTERNATIONAL. PAGE 5141.
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

This dissertation presents <math> <f> <blkbd><ty><rm><ge><blkbd><mit><ge><ty>p</ty><ty>Hc</ty></ge> </mit></blkbd></ge></rm></ty></blkbd></f> </math>, a new **compiler** for Parallel Haskell (<italic>pH</italic>) with complete support for the entire language. <italic>pH</italic> blends the powerful features of a high-level, higher-order language with implicitly parallel, non-strict semantics. The result is a language that is easy to **use** but tough to compile. The principal contribution of this work is a new code-generation algorithm that works directly on λ*, a terse intermediate representation based on the λ-calculus. All the power of the original language is **preserved** in λ*, and in addition, the new representation makes it easy to express the important concept of threads, groups of expressions that can **execute** sequentially, at a high-level.

Code generation proceeds in two steps. First, the **compiler** turns λ* programs into multithreaded code for the Shared- **Memory** Threaded (SMT) **machine** , **code** -generator, by providing special mechanisms for parallelism, and improves its portability, by isolating it from any particular SMP. Unlike previous **compilers** for similar languages, the SMT code **generated** by <math> <f> <blkbd><ty><rm><ge><blkbd><mit><ge><ty>p</ty><ty>Hc</ty></ge> </mit></blkbd></ge></rm></ty></blkbd></f> </math> makes **use** of suspensive threads for better **use** of sequential **processors** . Second, the **compiler** transforms SMT **instructions** into **executable** code. In **keeping** with the goals of simplicity and portability, the final product is emitted as a C program, with some extensions for **compiler** -controlled multithreading. This code is linked with a Run-Time System (RTS) that provides two facilities: a work-stealing scheduler and a **memory** manager with garbage-collection.

To evaluate the performance of the code, we **generated** instrumented programs. **Using** a small but varied set of benchmarks, we measured the components of run-time, the **instruction** mix, the scalability of code up to eight **processors** , and as a comparison, the single- **processor** performance against two of the latest Haskell **compilers** . We found encouraging speedups up to four **processors** , but rarely beyond. Moreover, we found substantial overhead in single- **processor** performance. These results are consistent with our emphasis on completeness and correctness first and on performance second. We conclude with some suggestions for future work that will remedy some of the shortcomings discovered in the evaluation. ( **Copies** available exclusively from MIT Libraries, Rm. 14-0551, Cambridge, MA 02139-4307. Ph. 617-253-5668; Fax 617-253-1690.)

**25/5/15 (Item 2 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


01647214 ORDER NO: AAD98-33480
**STATIC ASSIGNMENT OF MULTITHREADED SYSTEMS (PARTITIONING, MERGING)**
Author: LUNDY, JOE CONNOR
Degree: PH.D.
Year: 1998
Corporate Source/Institution: CLEMSON UNIVERSITY (0050)
Adviser: HAROLD GROSSMAN
Source: VOLUME 59/05-B OF DISSERTATION ABSTRACTS INTERNATIONAL. PAGE 2290. 137 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

This paper presents the design **implementation** of program translation and runtime system that will allow programs written in an imperative language to be translated into a multithreaded format and to be **executed** on a general-purpose multiprocessor system. Program translation techniques are **developed** that merge program **statements** into threads, and then partition these threads to the available **processors** . Both the merging and partitioning techniques **use** metrics that estimate the speedup that would result from performing the indicated merge or partition.

After the thread partitioning, the **source code** associated with the set of threads is compiled by a traditional **compiler** , and the resulting **machine code** is then **executed** by a shared- **memory** multiprocessor. During **execution** this multiprocess is never aware of the threaded-structure of the program. However, the sequence of **executing** the threads is completely controlled by static-dataflow processing techniques. This paper also includes the empirical results of testing this design and **implementation** , with a suite of eleven test programs.

**25/5/16 (Item 3 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


01428622 ORDER NO: AADAA-I9527657
**LOOP OPTIMIZATION TECHNIQUES ON MULTI-ISSUE ARCHITECTURES**
Author: KAISER, DAN RICHARD
Degree: PH.D.
Year: 1995
Corporate Source/Institution: THE UNIVERSITY OF MICHIGAN (0127)
Chair: TREVOR N. MUDGE
Source: VOLUME 56/04-B OF DISSERTATION ABSTRACTS INTERNATIONAL. PAGE 2136. 183 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

This work examines the interaction of **compiler** scheduling techniques with **processor** features such as the **instruction** issue policy. Scheduling techniques designed to exploit **instruction** level parallelism are **employed** to schedule **instructions** for a set of multi-issue architectures. A **compiler** is **developed** which supports block scheduling, loop unrolling, and software pipelining for a range of target architectures. The **compiler** supports aggressive loop optimizations such as induction variable detection and strength reduction, and code hoisting. A set of machine configurations based on the MIPS R3000 **ISA** are simulated, allowing the performance of the combined **compiler - processor** to be studied. The Aurora III, a prototype superscalar **processor** , is used as a case study for the interaction of **compiler** scheduling techniques with **processor** architecture.

Our results show that the scheduling technique chosen for the **compiler** has a significant impact on the overall system performance and can even change the rank ordering when comparing the performance of VLIW, DAE and superscalar architectures. Our results further show that, while significant, the performance effects of the **instruction** issue policy may not be as large as the effects of other **processor** features, which may be less costly to **implement** , such as 64 bit wide data paths or **store** buffers.

**25/5/17 (Item 4 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


01419238 ORDER NO: AADAA-I9519447
**THE FUNCTIONAL** MEMORY **APPROACH TO THE DESIGN OF CUSTOM COMPUTING MACHINES (MICROPROGRAMMING)**
Author: HALVERSON, RICHARD PEYTON, JR.
Degree: PH.D.
Year: 1994

Corporate Source/Institution: UNIVERSITY OF HAWAII (0085)
Chairperson: ART LEW
Source: VOLUME 56/02-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 919. 204 PAGES
Descriptors: **COMPUTER** SCIENCE; ENGINEERING, ELECTRONICS AND ELECTRICAL
Descriptor Codes: 0984; 0544

This dissertation describes a software system and related hardware architecture in which high level language programs are compiled into gate level logic circuitry that is configured specifically to **execute** the compiled program. A system whose **processor** can be dynamically reconfigured to suit different applications is known as a custom computing machine (CCM). We have designed a new class of CCMs based on the concept of functional **memory** (FM), which we construct by connecting field programmable gate arrays (FPGAs) in parallel with conventional random access **memory** (RAM). FM is used by the **processor** for computing the (possibly multi-operand) expressions of the high level language program in the combinational logic provided by the FPGAs. When all program expressions are computed in FM, the necessary **processor** **instruction** set reduces to a minimal number of moves and jumps.

Our functional **memory** **computer** (FMC) is a four FPGA FM prototype with a fifth FPGA programmed as the minimal **processor** . The language we adopted as the high level source language for programming the FMC is a decision-table (DT) variation of standard Pascal. DT programs for a shortest path and two sorting algorithms were translated, **executed** , and analyzed on the FMC. The second sorting program demonstrated a nondeterministic array selection function. An analysis for the shortest path program showed that **memory** load/ **store** counts remained comparable for FMC and von Neumann **implementations** . However, with the FMC, a 35% reduction in total **execution** steps occurred because all computation steps are performed in parallel on the FMC.

The problem of compiling high level DTs to low level FMC **object** **code** is more complex than for conventional machines because each single expression in the source program can translate into several tens of lines of FPGA circuit definition code. The Windows based system **developed** for this purpose includes a **compiler** that translates source programs into intermediate assembly language modules, and an operating system that invokes system routines for assembling, linking, placing and routing, and loading the FPGA machine level **object** **code** into the minimal **processor** and functional **memory** .

**25/5/18** **(Item 5 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


01279386 ORDER NO: AAD92-20752
**ABSTRACT INTERPRETATIONS AND ABSTRACT MACHINES: CONTRIBUTIONS TO A METHODOLOGY FOR THE** IMPLEMENTATION **OF LOGIC PROGRAMS**
Author: NILSSON, ULF
Degree: PH.D.
Year: 1992
Corporate Source/Institution: UNIVERSITETET I LINKOPING (SWEDEN) (0720)
Source: VOLUME 53/11-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 5827. 177 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

Because of the conceptual gap between high-level programming languages like logic programming and existing hardware, the problem of compilation often becomes quite hard. This thesis addresses two ways of narrowing this gap--program analysis through abstract interpretation and the introduction of intermediate languages and abstract machines. By means of abstract interpretations it is possible to infer program properties which are not explicitly present in the program--properties which can be used by a **compiler** to **generate** specialized code. We describe a framework for constructing and computing abstract interpretations of logic programs with

equality. The core of the framework is an abstract interpretation called the base interpretation which provides a model of the run-time behaviour of the program. The model characterized by the base interpretation consists of the set of all reachable computation states of a transition system specifying an operational semantics reminiscent of SLD-resolution. This model is in general not effectively computable, however, the base interpretation can be used for constructing new abstract interpretations which approximate this model. Our base interpretation combines both a simple and concise formulation with the ability of inferring a wide range of program properties. In addition the framework also supports efficient computing of approximate models **using** a chaotic iteration strategy. However, the framework supports also other computation strategies.

We also show that abstract interpretations may form a basis for **implementation** of deductive databases. We relate the magic templates approach to bottom-up evaluation of deductive databases with the base interpretation of C. Mellish and prove that they not only specify isomorphic models but also that the computations which lead up to those models are isomorphic. This implies that methods (for instance, evaluation and transformation techniques) which are applicable in one of the fields are also applicable in the other. As a side-effect we are also able to relate so-called "top-down" and "bottom-up" abstract interpretations.

Abstract machines and intermediate languages are often used to bridge the conceptual gap between language and hardware. Unfortunately--because of the way they are presented--it is often difficult to see the relationship between the high-level and intermediate language. In the final part of the thesis we propose a methodology for designing abstract machines of logic programming languages in such a way that much of the relationship is **preserved** all through the process. **Using** partial deduction and other transformation techniques a source program and an interpreter are "compiled" into a new program consisting of " **machine code** " for the source program and an abstract machine for the **machine code** . Based upon the appearance of the abstract machine the user may choose to modify the interpreter and repeat the process until the abstract machine reaches a suitable level of abstraction. We demonstrate how these techniques can be **applied** to derive several of the control **instructions** of Warren's Abstract Machine, thus complementing previous work by P. Kursawe who reconstructed several of the unification **instructions** **using** similar techniques.

**25/5/19 (Item 6 from file: 35)**


01204812 ORDER NO: AAD92-05881
**PERFORMANCE PREDICTION OF SUPERCOMPUTER PROGRAMS ( CACHE HIT RATIO, PARALLEL PROCESS)**
Author: ATAPATTU, DAYANANDA WIJESINGHE
Degree: PH.D.
Year: 1991
Corporate Source/Institution: INDIANA UNIVERSITY (0093)
Adviser: DENNIS B. GANNON
Source: VOLUME 52/09-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 4824. 154 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

The dissertation discusses static performance prediction techniques for programs on shared- **memory** multiprocessing systems. While static prediction of program flow is impossible in general, structured loops usually **execute** a statically known **instruction** sequence. In most scientific applications these nested loops dominate performance characteristics. Estimation of program performance, even when program flow is known, presents some problems. In a shared- **memory** system, program performance estimation is complicated due to the following factors: (1) The **compiler** may restructure source programs and the user has to understand the restructured **object code** . (2) There are facility conflicts as

various **instructions** compete for limited machine resources. (3) In a hierarchical **memory** system the time for a **memory** reference depends on the location of the data item. (4) In most shared- **memory** systems **memory** bandwidth **creates** a bottleneck which limits the data transfer rate.

The dissertation investigates performance prediction in the presence of the above problems. The theories presented are **implemented** in an interactive performance predictor and estimated values are compared against actual run-time results.

**25/5/20 (Item 7 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


01099603 ORDER NO: NOT AVAILABLE FROM UNIVERSITY MICROFILMS INT'L.
**AN EARLY** IMPLEMENTATION **OF REVISED ALGOL 68 GARBAGE COLLECTION** USING **RECOGNIZABLE POINTERS**
Author: SCHLICHTING, JOSEPHUS JOHANNES FRANCISCUS MARIA
Degree: DR.
Year: 1989
Corporate Source/Institution: TECHNISCHE UNIVERSITEIT TE DELFT (THE NETHERLANDS) (0951)
Source: VOLUME 51/02-C OF DISSERTATION ABSTRACTS INTERNATIONAL. PAGE 301. 150 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984
Location of Reference Copy: DELFT UNIVERSITY OF TECHNOLOGY, DELFT, THE NETHERLANDS

An almost complete **implementation** of ALGOL 68 Revised has been realized. The extensions to the language include removal of most restrictions imposed by the ALGOL 68 scope rule, separate compilation facilities, a method to allow the definition of operators in terms of machine **instructions** , called ICF-macros, and a FORTRAN interface.

Preludes and postludes may be nested, but each level of nesting requires compilation of the innermost level prelude and postlude.

The treatment of the scope rule restricts run-time scope checks to the checking of the scope of a routine at the time the routine is activated.

The **compiler** **executes** six complete scans of the source program in some intermediate form. The first three scans are largely machine independent and perform the lexical, syntactic and semantic analysis of the source text. The remaining scans **generate** , optimize and edit the **object** **code** .

The run-time system **employs** a stack and a heap to **store** data. The stack contains fixed size segments for each active procedure and all data that cannot be allocated in fixed segments or that is referred to by pointers, of which the scope is not known at compile time, is allocated on the heap.

This approach simplifies stack handling at the expense of heap activity. Therefore a simple and fast garbage collector is required. Pointers are internally represented as out of range floating point numbers in a format acceptable to but not **generated** by the floating point hardware **instructions** .

This representation of pointers simplifies the recognition of pointers in the marking phase of the garbage collector and also makes it possible to allocate the links of marked pointer chains in the pointers themselves.

The garbage collector based on this representation is simple and fast and does not require any special provisions in the **generated** code other than initialization of all data to a bit pattern that is distinguishable from the patterns used to represent pointers.

By exploiting the hardware characteristics of the Control Data **computer** system we succeeded in building a fast and elegant garbage collector without placing a burden on the **compiler** .

**25/5/21 (Item 8 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online

1080854 ORDER NO: AAD89-24956
**SELF-SCHEDULING, DATA SYNCHRONIZATION AND PROGRAM TRANSFORMATION FOR MULTIPROCESSOR SYSTEMS**
Author: TANG, PEIYI
Degree: PH.D.
Year: 1989
Corporate Source/Institution: UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN (0090)
ADVISER: PEN-CHUNG YEW
Source: VOLUME 50/07-B OF DISSERTATION ABSTRACTS INTERNATIONAL. PAGE 3023. 230 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

The limitation of vector supercomputing and of device speed has led to the **development** of multiprocessor supercomputers. Although large tightly-coupled shared- **memory** multiprocessor systems have become feasible, such systems cannot be considered successful, unless their numerous **processors** can coordinate with each other efficiently for a wide range of applications. Advanced software techniques and their supports from system architectures are keys to the success of modern multiprocessor supercomputers.

This dissertation first concentrates on two important software issues for large multiprocessor systems: **processor** scheduling and data synchronization.

Self-scheduling, an efficient dynamic heuristic scheduling, is a practical solution to the scheduling problem of multiprocessor systems. We propose several self-scheduling schemes which can be used by a **compiler** to **generate** self-scheduling **object codes** for parallel programs. Since busy-waiting is used to enforce cross-iteration data dependences, deadlocks in self-scheduling are possible. We identify the conditions that allow deadlock-free self-scheduling for different self-scheduling models and propose the **use** of an appropriate scheduling order for allocating **processors** to prevent deadlocks. Self-scheduling order also has significant impact on the performance of parallel loops with cross-iteration data dependences. We propose the shortest-delay self-scheduling (SDSS) order based on Doacross delays determined by cross-iteration data dependences. We show by simulation that SDSS can offer near-optimal performance in most cases. A compile-time program transformation for SDSS is also presented.

Data synchronization is necessary to enforce cross-iteration data data dependences. We propose a set of data-level synchronization **instructions** to support data synchronization. **Compiler** algorithms for generating these data-level synchronization **instructions** for different types of subscript functions are presented.

The last part of the dissertation addresses an architecture issue of large multiprocessor systems: "hot-spot" contention. We suggest the **use** of software combining to distribute "hot-spot" addressings. A number of software combining algorithms for different access patterns, such as barrier synchronization, fetch-and-add type of operations and semaphore P/V operations, are presented.

**25/5/22 (Item 9 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


1068559 ORDER NO: AAD89-16082
**OPTIMIZATION AND GARBAGE COLLECTION IN ADA PROGRAMS ON SHARED** MEMORY COMPUTERS
Author: OPEROWSKY, HOWARD LAWRENCE
Degree: PH.D.
Year: 1989
Corporate Source/Institution: NEW YORK UNIVERSITY (0146)
ADVISER: EDMOND SCHONBERG

Source: VOLUME 50/04-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 1509. 295 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

**Compiler development** for Ada is still in its infancy. Despite its goal of supporting embedded systems in an efficient manner, Ada programs still tend to be large and slow. In this thesis, we investigate three issues related to the efficient **implementation** of Ada programs: run-time representation of types and objects, reduction of run-time constraint checking, and parallel garbage collection on a shared **memory** multiprocessor.

We present a collection of type templates for scalar and composite types which are storage-efficient and allow for efficient **object code** to be **produced** by the code generator. We present an algorithm for constructing these templates at run-time when constraint information is unavailable at compile-time.

We show that a global optimizer is not required to reduce the overhead of constraint checking in Ada programs. We present a series of data-flow equations for available expressions and **use** them as the basis for a simple algorithm to eliminate redundant constraint checks. The algorithm is syntax-directed and is **executed** in a single pass over the source program's abstract syntax tree. No control flow analysis is required. Our algorithm also includes constant propagation **using** an extended framework and induction variable analysis. Because the algorithm operates on the abstract syntax tree, induction variable analysis is simplified. Although programs with goto **statements** are not considered, the exit **statement** is handled fully. We also examine the effects of shared variables and exception handling.

No commercial **compiler** for Ada currently performs garbage collection. We examine the difficulties in garbage collection presented by Ada and present practical algorithms for Ada on shared **memory** multiprocessors. We extend Kung and Song's on-the-fly garbage collection algorithm to support multiple tasks on the NYU Ultracomputer/IBM RP3 **computers** . We prove that no additional synchronization is required because of Ada's rules on the **use** of shared variables.

**25/5/23 (Item 10 from file: 35)**
DIALOG(R)File 35:Dissertation Abs Online


1046085 ORDER NO: AAD89-05875
**DENOTATIONAL SEMANTICS-DIRECTED** COMPILER **GENERATION FOR DATA FLOW MACHINES**
Author: CHAO, WILLIAM SHAN-JON
Degree: PH.D.
Year: 1988
Corporate Source/Institution: THE UNIVERSITY OF ALABAMA IN BIRMINGHAM ( 0005)
CHAIRMAN: BARRETT R. BRYANT
Source: VOLUME 49/12-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 5395. 135 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

The denotational semantics approach, also known as the mathematical approach, to formal specification of programming languages contains detailed information for the **implementation** of the languages. A **compiler** for a programming language can be systematically constructed from the formal denotational semantics specification by applying the function of the specification to the source program. This is called the compile evaluate approach for denotational semantics or **compiler** generation.

There are two serious problems which arise in existing denotational semantics **implementation** systems. The first one is that the semantics of the **generated** code may be different from the semantics of the source program. This semantics inconsistency occurs because the denotational

semantics of the programming language is not computationally correct and the **compiler** computation steps are not properly evaluated during code optimization of the **generated** denotational expression. The second problem is that the **generated** code may not be as efficient as target code **generated** by a conventional hand-coded **compiler** due to the fact that most denotational semantic **implementation** systems **generate** theoretical instead of real **machine code** .

Our **compiler** generation system is an improvement over other **compiler** generation systems in three ways: (1) We define a computationally correct denotational semantics for a programming language based on correct order computation rules. Then we do all partial evaluation based on the computation rule which are explicitly specified in the definitions. (2) We show how denotational code optimization of imperative programs can be taken much further than present denotational semantics-based methods by unfreezing the **store** algebra and allowing partial evaluation to proceed on **store** -based entities. We have specifically demonstrated constant folding and in-line function expansion **using** this technique. Additional optimizations we perform are the detection of loop invariants and elimination of common subexpressions in denotational expressions. All these optimizations make considerable improvements in the denotational **machine code** . (3) We introduce a set of combinators for the optimized denotational expressions which correspond to the **instructions** of a data flow machine. These combinators complete a formal mapping from the source program to data flow **machine code** which displays the maximum amount of parallelism of the original imperative program.

The result of our system is that an imperative program can be compiled into highly optimized code which may be **executed** in parallel for maximum efficiency.

**25/5/24 (Item 11 from file: 35)**


771855 ORDER NO: AAD82-05047
**OPTIMAL MIXED CODE GENERATION FOR** MICROCOMPUTERS
Author: PHOTOPOULOS, DEMETRIUS S.
Degree: PH.D.
Year: 1981
Corporate Source/Institution: NORTHEASTERN UNIVERSITY (0160)
Source: VOLUME 42/09-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 3752. 219 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

A method of generating the most effective partially assembled versions of interpretive programs is introduced.

In the first part an interpretive language, XIL, is presented which is quite efficient in its **use** of **memory** space. Emphasis is given to the language facilities, that make possible the mixing of interpretive code and **machine code** in the same program. The interpreter, and the **assembler** that translates sections of XIL programs to VAX-11/780 **machine code** , are described.

In the second part the algorithms are **developed** , which are for deriving the most effective partially assembled program versions. The algorithms work based on the frequency of **execution** of the **statements** of the programs, and they take into account the effects, in time and space **utilization** , of the special **instructions** which accomplish the transitions from one coding mode to the other. An example of application of the algorithms is given, where the **execution** -time versus **memory** -space tradeoffs are found, for the most effective partially assembled versions of a XIL program, that translates PASCAL programs to XIL code.

**25/5/25 (Item 12 from file: 35)**

746827 ORDER NO: AAD81-10908

# A COMPUTER SYSTEM FOR CHECKING PROOFS

Author: JOHNSON, SCOTT DAVID

Degree: PH.D.
Year: 1981
Corporate Source/Institution: CORNELL UNIVERSITY (0058)
Source: VOLUME 41/12-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 4579. 345 PAGES
Descriptors: COMPUTER SCIENCE
Descriptor Codes: 0984

In 1975, a project was begun at Cornell University to **produce** a programming logic in which formal proofs of programs could be written and mechanically checked. The result of this project is the programming logic PL/CV. The PL/CV logic contains rules for the first order predicate calculus, Peano arithmetic, the theory of equality, and the PL/CS programming language. The thesis concerns a **computer** system used to check proofs written in PL/CV. The correctness of the overall design of the Proof Checker and several of the algorithms **employed** are argued.

The Proof Checker is divided into two modules. The first module is the Local Context Checker (LCC), which inputs a proof and processes it in much the same fashion as a **compiler** processes a program. But instead of producing **machine code**, the LCC **produces** a series of **instructions** called V-code. The LCC performs checking that only requires examining a small window in the proofs. The global aspects of the proof are represented by the V-code, and are checked by the second module, the V-code Interpreter. Abbreviation rules are built into the LCC and automatic deduction rules are built into the V-code Interpreter that reduce the amount of detail required in proofs.

It is important that the Proof Checker have a good method for representing quantified formulas. A method of representation is presented that eliminates some technical problems often associated with bound variables and substitution. Bound variables are represented by **storing** the number of quantifiers along the path in the parse tree from the variable to the binding quantifier in lieu of a variable name. Operations that must be commonly performed on quantified formulas by the LCC can be efficiently performed **using** this representation.

The PL/CV predicate calculus rules are proved to be equivalent to Gentzen's Natural Deduction. The syntax of the PL/CV predicate calculus and its macro feature are discussed from the standpoint of human engineering. The **use** of ALGOL scope rules and V-code in checking predicate calculus proofs is discussed.

For each construct in the programming language PL/CS, the proof rule used in PL/CV is presented and its strengths and weaknesses are discussed. Then the V-code **generated** for that construct is presented. A theorem is proved that is used to derive the V-code **generated** for each of the control constructs. A generalized control construct is proposed as a replacement for all but one of the control constructs in PL/CS.

The V-code Interpreter is based on two algorithms that solve similar problems. In each case, we are given a set S of pairs of formulas that are known to be equivalent. If P(,1) and P(,2) are formulas, then P(,1) = P(,2) is defined to hold whenever it is possible to transform P(,1) to P(,2) by applying to P(,1) a sequence of substitutions resulting in P(,2), where each substitution replaces an occurrence of some Q(,1) by a Q(,2) such that (Q(,1), Q(,2)) or (Q(,2), Q(,1)) is a pair in S. The problem is to compute the relation (TBOND).

One solution is given for a particular infinite set S, which includes axioms such as commutativity and associativity. Given two formulas P and Q, it can be determined in O((VBAR)P(VBAR)('2) + (VBAR)Q(VBAR)('2)) time whether P (TBOND) Q. A second solution is given for any finite set S. A series of **instructions** that adds pairs to S and tests for equivalences based on the current value of S can be processed in expected time O(N log N), where N is the total size of all the formulas in the **instructions**.

The thesis is concluded with a short essay on the direction the author feels future research in program verification should take.

25/5/26 (Item 13 from file: 35)
DIALOG(R)File 35:Dissertation Abs Online


742771 ORDER NO: AAD81-07880
A COMPUTER **ARCHITECTURE FOR THE DYNAMIC OPTIMIZATION OF HIGH-LEVEL LANGUAGE PROGRAMS**
Author: HARBISON, SAMUEL POLLOCK, III
Degree: PH.D.
Year: 1980
Corporate Source/Institution: CARNEGIE-MELLON UNIVERSITY (0041)
Source: VOLUME 41/10-B OF DISSERTATION ABSTRACTS INTERNATIONAL. PAGE 3831. 177 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

One of the disadvantages of high-level languages has been the necessity of **using** expensive optimizing **compilers** to **generate** efficient **object code** . This thesis describes and analyzes TM, a **computer** architecture that dynamically performs many optimizations commonly seen in such **compilers** , including common subexpression elimination, code motion, and register allocation. The architecture is based on an efficient, stack-oriented **instruction** set and is augmented with a special **cache** that holds the values of expressions and their dependencies. Experiments comparing the performance of TM and the a traditional register architecture on a set of test programs show that a simple, nonoptimizing **compiler** can **generate** **object** **code** for TM that is smaller and faster than the code **produced** by an optimizing **compiler** for the register architecture. A cost analysis shows that TM's hardware mechanisms need not be expensive in comparison with traditional architectures and that the savings in **compiler** **development** costs are significant. Additional experiments investigate **cache** behavior and analyze alternative **implementations** .

25/5/27 (Item 14 from file: 35)
DIALOG(R)File 35:Dissertation Abs Online


699278 ORDER NO: AAD80-24756
**A LANGUAGE-ORIENTED APPROACH TO** COMPUTER **ARCHITECTURE**
Author: WILCOX, CLARK ROGER
Degree: PH.D.
Year: 1980
Corporate Source/Institution: STANFORD UNIVERSITY (0212)
Source: VOLUME 41/05-B OF DISSERTATION ABSTRACTS INTERNATIONAL. PAGE 1835. 380 PAGES
Descriptors: **COMPUTER** SCIENCE
Descriptor Codes: 0984

A language-oriented approach to **computer** architecture starts with a high-level language, from which an "ideal" program representation and **execution** environment is derived, which in turn determines an "ideal" **processor** architecture. This is in contrast to the conventional approach wherein a single fixed **instruction** set supports all languages.

In order to examine and evaluate the issues raised by the language-oriented approach, a methodology is presented for deriving successive levels in a design hierarchy consisting of the following components: an abstract syntax and abstract semantics for the language; derivation and interpretation of a token stream and a bit-oriented code stream; **language** **processor** architecture; and program monitoring facilities.

The methodology is **applied** to a particular machine-independent

high-level language in order to obtain detailed data concerning static program size and dynamic **execution** characteristics. A language-specific representation, postfix code, is designed according to this methodology. Postfix code is up to one fifth the size of conventional **machine code** and is simple to **generate**.

A **computer** architecture is designed for interpretive **execution** of bit-oriented code streams. The 32-bit **processor** has a control **store** for micro-coded interpreters, hardware maintenance of the code stream, table-driven variable-width field extraction and operator dispatch in parallel with micro-code **execution**, and two micro-level stacks. The **processor** is unbiased with regard to the source language in that it defines neither the form of the **instruction** stream, nor the formats of data structures. A micro-coded postfix interpreter is used to **execute** a number of program modules compiled into postfix code. The postfix **execution** is compared with a conventional **implementation** and shown to be superior in every measured quantity; for example it has half as many **instruction** loads.

A novel feature of postfix code is its ability to be decompiled into source text, including variable names. This provides the basis for a display-oriented program **development** and monitoring environment which can decompile and display the program text during a debugging session, with the cursor following the **execution** locus while in single-step mode. Breakpoints are set by moving the cursor to the desired point in the displayed text.

**25/5/28 (Item 1 from file: 2)**



6735609 INSPEC Abstract Number: C2000-11-5220P-042

**Title: Improving binary compatibility in VLIW machines through** compiler **assisted dynamic rescheduling**

Author(s): Biglari-Abhari, M.; Eshraghian, K.; Liebelt, M.J.
Author Affiliation: Centre for Very High Speed Microelectron. Syst., Edith Cowan Univ., Joondalup, WA, Australia

Conference Title: Proceedings of the 26th Euromicro Conference. EUROMICRO 2000. Informatics: Inventing the Future Part vol.1 p.386-93 vol.1
Editor(s): Vajda, F.
Publisher: IEEE Comput. Soc, Los Alamitos, CA, USA
Publication Date: 2000 Country of Publication: USA 2 vol. xv+467+475 pp.
ISBN: 0 7695 0780 8 Material Identity Number: XX-2000-02110
U.S. Copyright Clearance Center Code: 1089 6503/2000/$10.00
Conference Title: Proceedings of Euromicro Workshop on Multimedia and Telecommunications
Conference Date: 5-7 Sept. 2000 Conference Location: Maastricht, Netherlands
Language: English Document Type: Conference Paper (PA)
Treatment: Applications (A); Practical (P)

Abstract: One of the main problems that prevents extensive **use** of VLIW architectures for non-numeric programs is lack of **object code** (or binary) compatibility among different **implementations** of the same architecture. This is due to exposing all architectural features to **generate** code at compile time. New features of a VLIW machine may lead to incorrect results by **executing** the code compiled for the older machine. In this paper, a new approach to overcome this problem is presented, which we call dynamic VLIW generation (DVG). It is performed with the help of code annotation provided by the **compiler**, to reduce the complexity of the required hardware. In the DVG technique, operations are rescheduled for the new machine at the time of **instruction cache** miss repair. In this way, the rescheduler hardware is not located in the **execution** pipeline engine avoiding potentially longer cycle times. To simplify the dependency checking hardware, dependency information is encoded for each operation at compile time. This information can be combined into the final binary code, or may be provided as a separate file, which can be loaded into **memory** at

**execution** time by the OS loader. In this technique operations can be rescheduled speculatively and a mechanism is presented to prevent destroying the contents of live registers. Experimental results show that the performance of rescheduled code **using** the DVG technique is about 10% worse than code compiled directly for the target **processor** . (21 Refs)

Subfile: C

Descriptors: computational complexity; parallel architectures; performance evaluation; **processor** scheduling

Identifiers: binary compatibility; VLIW machines; **compiler** assisted dynamic rescheduling; VLIW architectures; dynamic VLIW generation; code annotation; **instruction cache** miss repair; **execution** pipeline engine

Class Codes: C5220P (Parallel architecture); C5440 (Multiprocessing systems); C4240C (Computational complexity); C5470 (Performance evaluation and testing)



**25/5/29 (Item 2 from file: 2)**



6450316 INSPEC Abstract Number: C2000-02-6150C-007

**Title**: Compiler **optimization by enhanced loop fusion**

Author(s): Iwasawa, K.

Author Affiliation: Dept. of Comput. Sci., Takushoku Univ., Tokyo, Japan

Conference Title: Proceedings of the Seventeenth IASTED International Conference. Applied Informatics p.634-7

Editor(s): Hamza, M.H.

Publisher: ACTA Press, Anaheim, CA, USA

Publication Date: 1999 Country of Publication: USA 699 pp.

ISBN: 0 88986 241 9 Material Identity Number: XX-1999-00795

Conference Title: Proceedings of 17th IASTED International Conference on Applied Informatics (AI'99)

Conference Sponsor: IASTED

Conference Date: 15-18 Feb. 1999 Conference Location: Innsbruck, Austria

Language: English Document Type: Conference Paper (PA)

Treatment: Practical (P)

Abstract: Loop fusion is often very effective in achieving run time speed up because it reduces the numbers of loop control **instructions** in a program. In particular, new programming languages like Fortran90/HPF include array descriptions, which allow operations that theoretically can be **executed** simultaneously on each array element; in practice, the **compiler** can choose how to **execute** these simultaneous operations. Vector **processors** can **execute** these simultaneous operations efficiently, but for a **processor** that does not have any vector facility, the **compiler** **generates** many small loops from array descriptions, and these loops result in poor performance. In such cases, if the **use** of loop fusion causes a change in the order of **statements** in the program, then the **compiler** has to **keep** track of the semantics of the program. This paper presents a **compiler** optimization technique for loop fusion of array descriptions; the technique is also useful for the for-loop of the C language. The motivation behind **developing** this technique is to optimize the speed of the **object code created** by loop fusion, and to clarify the method used by the **compiler** . It has been claimed that if the direction of data dependency were changed by loop fusion, it would be impossible to fuse loops. The technique described can fuse some such loops by **employing** temporary arrays. We believe that the technique is quite powerful and general. (4 Refs)

Subfile: C

Descriptors: optimising **compilers** ; subroutines

Identifiers: enhanced loop fusion; **compiler** optimization; run time speed up; loop control **instructions** ; array descriptions; Fortran90; HPF; Vector **processors** ; program **statements** ; program semantics; for-loop; C language; **object code** ; data dependency; temporary arrays

Class Codes: C6150C (Compilers, interpreters and other processors)

25/5/30 (Item 3 from file: 2)
DIALOG(R)File 2:INSPEC


6449160 INSPEC Abstract Number: B2000-02-1265F-007, C2000-02-5130-003
**Title: A memory power optimization technique for application specific embedded systems**
Author(s): Ishihara, T.; Yasuura, H.
Author Affiliation: Dept. of Comput. Sci. & Commun. Eng., Kyushu Univ., Fukuoka, Japan
Journal: IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences vol.E82-A, no.11 p.2366-74
Publisher: Inst. Electron. Inf. & Commun. Eng,
Publication Date: Nov. 1999 Country of Publication: Japan
CODEN: IFESEX ISSN: 0916-8508
SICI: 0916-8508(199911)E82A:11L.2366:MPOT;1-T
Material Identity Number: P710-1999-012
Language: English Document Type: Journal Paper (JP)
Treatment: Applications (A); Practical (P); Experimental (X)
Abstract: In this paper, a novel application specific power optimization technique **utilizing** small **instruction** ROM which is placed between an **instruction cache** or a main program **memory** and CPU core is proposed. Our optimization technique targets embedded systems which assume the following: (i) **instruction** memories are organized by two on-chip memories, a main program **memory** and a subprogram **memory** , (ii) these two memories can be independently powered-up or powered-down by a special **instruction** of a core **processor** , and (iii) a **compiler** optimizes an allocation of **object code** into these two memories so as to minimize average of read energy consumption. In many application programs, only a few basic blocks are frequently **executed** . Therefore, allocating these frequently **executed** basic blocks into low power subprogram **memory** leads significant energy reduction. Our experiments with actual ROM (Read Only **Memory** ) modules **created** with 0.5 mu m CMOS process technology, and MPEG2 codec program demonstrate significant energy reductions up to more than 50% at best case over the previous approach that **applies** only divided bit and word lines structure. (10 Refs)
Subfile: B C
Descriptors: application specific integrated circuits; circuit optimisation; CMOS digital integrated circuits; embedded systems; hardware-software codesign; **instruction** sets; low-power electronics; **microprocessor** chips
Identifiers: **memory** power optimization technique; application specific embedded systems; small **instruction** ROM; **instruction** memories; core **processor** ; **object code** ; read energy consumption; low power subprogram **memory** ; CMOS process technology; energy reductions; 0.5 micron
Class Codes: B1265F (Microprocessors and microcomputers); B1265A (Digital circuit design, modelling and testing); B2570D (CMOS integrated circuits); C5130 (Microprocessor chips); C7410D (Electronic engineering computing); C5215 (Hardware-software codesign)
Numerical Indexing: size 5.0E-07 m


25/5/31 (Item 4 from file: 2)
DIALOG(R)File 2:INSPEC


6449156 INSPEC Abstract Number: B2000-02-1265F-005, C2000-02-5135-004
**Title: A hardware/software cosynthesis system for digital signal processor cores**
Author(s): Togawa, N.; Yanagisawa, M.; Ohtsuki, T.
Author Affiliation: Dept. of Electron. Inf. & Commun. Eng., Waseda Univ., Tokyo, Japan
Journal: IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences vol.E82-A, no.11 p.2325-37

Publisher: Inst. Electron. Inf. & Commun. Eng,
Publication Date: Nov. 1999 Country of Publication: Japan
CODEN: IFESEX ISSN: 0916-8508
SICI: 0916-8508(199911)E82A:11L.2325:HSCS;1-B
Material Identity Number: P710-1999-012
Language: English Document Type: Journal Paper (JP)
Treatment: Applications (A); Practical (P); Experimental (X)
Abstract: This paper proposes a hardware/software cosynthesis system for digital signal **processor** cores and a hardware/software partitioning algorithm which is one of the key issues for the system. The target **processor** has a VLIW-type core which can be composed of a **processor** kernel, multiple data **memory** buses (X-bus and Y-bus), hardware loop units, addressing units, and multiple functional units. The **processor** kernel includes five pipeline stages (RISC-type kernel) or three pipeline stages (DSP-type kernel). Given an application program written in the C language and a set of application data, the system synthesizes a **processor** core by selecting an appropriate kernel (RISC-type or DSP-type kernel) and required hardware units according to the application program/data and the hardware costs. The system also **generates** the **object code** for the application program and a software environment ( **compiler** and simulator) for the **processor** core. The experimental results demonstrate that the system synthesizes **processor** cores effectively according to the features of an application program and the synthesized **processor** cores **execute** most application programs with the minimum number of clock cycles compared with several existing **processors** . (36 Refs)
Subfile: B C
Descriptors: C language; digital signal processing chips; hardware-software codesign; parallel architectures; pipeline processing; reduced **instruction** set computing
Identifiers: hardware/software cosynthesis system; digital signal **processor** cores; hardware/software partitioning algorithm; **processor** kernel; multiple data **memory** buses; hardware loop units; addressing units ; multiple functional units; pipeline stages; C language; RISC-type kernel; DSP-type kernel; hardware costs; **object code** ; software environment; application program; application programs; clock cycles
Class Codes: B1265F (Microprocessors and microcomputers); B1265A (Digital circuit design, modelling and testing); C5135 (Digital signal processing chips); C5215 (Hardware-software codesign); C7410D (Electronic engineering computing); C5220P (Parallel architecture)


**25/5/32 (Item 5 from file: 2)**
DIALOG(R)File 2:INSPEC


6084624 INSPEC Abstract Number: C9812-6150C-034
**Title: Efficient JavaVM just-in-time compilation**
Author(s): Krall, A.
Author Affiliation: Inst. fur Computersprachen, Tech. Univ. Wien, Austria
Conference Title: Proceedings. 1998 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.98EX192) p.205-12
Publisher: IEEE Comput. Soc, Los Alamitos, CA, USA
Publication Date: 1998 Country of Publication: USA xiii+435 pp.
ISBN: 0 8186 8591 3 Material Identity Number: XX98-02839
U.S. Copyright Clearance Center Code: 0 8186 8591 3/98/$10.00
Conference Title: Proceedings 1998 International Conference on Parallel Architectures and Compilation Techniques
Conference Sponsor: IFIP WG 10.3; IEEE Comput. Soc.; INRIA; ACM; TELECOM Paris
Conference Date: 12-18 Oct. 1998 Conference Location: Paris, France
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P)
Abstract: Conventional **compilers** are designed for producing highly optimized code without paying much attention to compile time. The design goals of Java just-in-time **compilers** are different: **produce** fast code at the smallest possible compile time. In this article we present a very

fast algorithm for translating JavaVM byte code to high quality **machine code** for RISC **processors**. This algorithm handles combines **instructions**, does **copy** elimination and coalescing and does register allocation. It comprises three passes: basic block determination, stack analysis and register preallocation, final register allocation and **machine code** generation. This algorithm replaces an older one in the CACAO JavaVM **implementation** reducing the compile time by a factor of seven and producing slightly faster **machine code**. The speedup comes mainly from following simplifications: fixed assignment of registers at basic block boundaries, simple register allocator better exception handling better **memory** management and fine tuning the **implementation**. The CACAO system is currently faster than every JavaVM **implementation** for the Alpha **processor** and **generates machine code** for all used methods of the javac **compiler** and its libraries in 60 milliseconds on an Alpha workstation. (16 Refs)

Subfile: C

Descriptors: parallel programming; program **compilers**

Identifiers: **compilers**; JavaVM; RISC **processors**; register allocation; block determination; stack analysis; register preallocation; CACAO system; JavaVM **implementation**; native code compilation; just-in-time compilation

Class Codes: C6150C (Compilers, interpreters and other processors); C6110P (Parallel programming).



**25/5/33 (Item 6 from file: 2)**

DIALOG(R)File 2:INSPEC



5745268 INSPEC Abstract Number: C9712-6150C-008

**Title: An** object code **compression approach to embedded** processors

Author(s): Yoshida, Y.; Song, B.-Y.; Okuhata, H.; Onoye, T.; Shirakawa, I.

Author Affiliation: Dept. Inf. Syst. Eng., Osaka Univ., Japan

Conference Title: Proceedings 1997 International Symposium on Low Power Electronics and Design (IEEE Cat. No.97TH8332) p.265-8

Publisher: ACM, New York, NY, USA

Publication Date: 1997 Country of Publication: USA x+335 pp.

ISBN: 0 89791 903 3 Material Identity Number: XX97-02049

U.S. Copyright Clearance Center Code: 0 89791 903 3/97/08..$3.50

Conference Title: Proceedings of 1997 International Symposium on Low Power Electronics and Design

Conference Sponsor: ACM SIGDA; IEEE Circuits & Syst. Soc

Conference Date: 18-20 Aug. 1997 Conference Location: Monterey, CA, USA

Language: English Document Type: Conference Paper (PA)

Treatment: Applications (A); Practical (P); Experimental (X)

Abstract: A low-power **processor** architecture is described dedicatedly for embedded application programs by means of an **object code** compression approach. This approach unifies duplicated **instructions** existing in the embedded program and assigns a compressed **object code** to such an **instruction**. An **instruction** decompressor is constructed so as to **generate** an **object code** from each compressed **object code** (pseudo code) input. A single-chip **implementation** of this decompressor together with a **processor** core can effectively reduce the bandwidth required for the I/O interface. To demonstrate the practicability of the proposed approach, experiments are **applied** to an embedded **processor** ARM610 which attains 62.5% code compression, and hence 42.3% of the power consumption of **instruction memory** can be reduced. (12 Refs)

Subfile: C

Descriptors: data compression; **microprocessor** chips; program **compilers**; real-time systems

Identifiers: **object code** compression; embedded **processors**; low-power **processor** architecture; embedded application programs; **instruction** decompressor; single-chip **implementation**; I/O interface bandwidth requirement; ARM610 **processor**; power consumption reduction; **instruction memory**

Class Codes: C6150C (Compilers, interpreters and other processors); C5130 (Microprocessor chips)
Copyright 1997, IEE

**25/5/34 (Item 7 from file: 2)**
DIALOG(R)File 2:INSPEC


5513190 INSPEC Abstract Number: C9704-6150C-024
**Title: Highlights from nhc-a space-efficient Haskell** compiler
Author(s): Rojerno, N.
Author Affiliation: Dept. of Comput. Sci., Chalmers Univ. of Technol., Goteborg, Sweden
Conference Title: Conference Record of FPCA '95. SIGPLAN-SIGARCH-WG2.8. Conference on Functional Programming Languages and Computer Architecture p.282-92
Publisher: ACM, New York, NY, USA
Publication Date: 1995 Country of Publication: USA viii+333 pp.
ISBN: 0 89791 719 7 Material Identity Number: XX95-01353
U.S. Copyright Clearance Center Code: 0 89791 719 7/95/0006.$3.50
Conference Title: Proceedings of 7th Annual SIGPLAN/SIGARCH/WG2.8 Conference on Functional Programming Languages and Computer Architecture
Conference Sponsor: ACM SIGPLAN; ACM SIGARCH; IFIP
Conference Date: 25-28 June 1995 Conference Location: La Jolla, CA, USA
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P)
Abstract: Self-compiling **implementations** of Haskell, i.e., those written in Haskell, have been and, except one, are still space consuming monsters. **Object code** size for the **compilers** themselves are 3-8 MByte, and they need 12-20 MByte to recompile themselves. One reason for the huge demands for **memory** is that the main goal for these **compilers** is to **produce** fast code. However, the **compiler** described in this paper, called nhc for Nearly a Haskell **Compiler** , is the above mentioned exception. This **compiler** concentrates on **keeping memory** usage down, even at a cost in time. The code **produced** is not fast but nhc is usable, and the resulting programs can be run on **computers** with small **memory** . This paper describes some of the **implementation** choices done in the Haskell part of the **source code** , to reduce **memory** consumption in nhc. It is possible to **use** these also in other Haskell **compilers** with no, or very small, changes to their run-time systems. Time is neither the main focus of nhc nor of this paper, but there is nevertheless a small section about the speed of nhc. The most notable observation concerning speed is that nhc spends approximately half the time processing interface files, which is much more than needed in the type checker. Processing interface files is also the most space consuming part of nhc in most cases. It is only when compiling source files with large sets of mutually recursive functions that more **memory** is needed to type check than to process interface files. (16 Refs)
Subfile: C
Descriptors: functional languages; functional programming; program **compilers** ; software performance evaluation
Identifiers: nhc; Haskell **compiler** ; self-compiling **implementations** ; **object code** size; **memory** demands; Nearly a Haskell **Compiler** ; functional language; **memory** consumption; run-time systems; speed; interface files; type checker; source files; mutually recursive functions; type checking; 8 MByte; 20 MByte
Class Codes: C6150C (Compilers, interpreters and other processors); C6110 (Systems analysis and programming); C6140D (High level languages)
Numerical Indexing: memory size 8.4E+06 Byte; memory size 2.1E+07 Byte
Copyright 1997, IEE

**25/5/35 (Item 8 from file: 2)**
DIALOG(R)File 2:INSPEC

5174931 INSPEC Abstract Number: C9603-6115-008
**Title: Programming PowerPC embedded applications**
Author(s): Mihalik, S.; Sobek, S.; Zucker, S.
Journal: Embedded Systems Programming vol.8, no.12 p.82-91
Publisher: Miller Freeman,
Publication Date: Dec. 1995 Country of Publication: USA
CODEN: EYPRE4 ISSN: 1040-3272
SICI: 1040-3272(199512)8:12L.82:PPEA;1-R
Material Identity Number: O692-96001
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: A common problem with **developing** embedded applications today is lack of interoperability. Software written for one vendor's tools cannot interoperate with tools from another vendor. Therefore, a **developer** cannot "mix and match" **compilers** , debuggers, and other software tools. In the absence of a common ABI, code libraries are nonportable, which eliminates the benefit of reusing **object code** . In addition, assembly language code must be rewritten when used with tools or other code having different register and **memory** conventions. These restrictions also affect tool vendors. For example, an outstanding debugging product that cannot interoperate with many **compilers** will have fewer sales. The PowerPC Embedded Application Binary Interface was **developed** by industry members who desired interoperability for embedded PowerPC applications. The EABI represents an optimization of the SVR4 desktop application Binary Interface for embedded applications. The embedded optimizations made were to minimize **memory** usage while maintaining performance. An Application Binary Interface (ABI) is a set of software conventions to allow software interoperability. For example, PowerPC **processors** do not have a hardware stack pointer. Consequently, software engineers working on a project must agree on which general purpose register to **use** as the stack pointer. In addition, any off-the-shelf operating system, **compiler** , or debugger used in the project must support **using** that register as a stack pointer, as well as other related conventions. What is the EABI? The PowerPC Embedded Application Binary Interface (EABI) is an extension of the industry standard Unix. (2 Refs)
Subfile: C
Descriptors: open systems; programming environments; real-time systems; reduced **instruction** set computing; Unix
Identifiers: PowerPC embedded applications; interoperability; **compilers** ; debuggers; software tools; code libraries; assembly language code; **memory** conventions; register conventions; tool vendors; PowerPC Embedded Application Binary Interface; embedded PowerPC applications; SVR4 desktop application binary interface; software interoperability; hardware stack pointer; general purpose register; Unix
Class Codes: C6115 (Programming support); C6150J (Operating systems); C6150N (Distributed systems software)


**25/5/36 (Item 9 from file: 2)**
DIALOG(R)File 2:INSPEC


5081615 INSPEC Abstract Number: C9512-6150C-001
**Title: Nanocomputer Optimizing Target** Compiler : **the** processor **-independent core**
Author(s): Hendtlass, T.
Journal: Forth Dimensions vol.17, no.3 p.20-31
Publication Date: Sept.-Oct. 1995 Country of Publication: USA
CODEN: FODMD5 ISSN: 0884-0822
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: The Nanocomputer Optimizing Target **Compiler** is a **compiler** shell which has been written to assist programming modern nanocomputers, small single chip **processors** with integrated RAM, ROM, and I/O. New nanocomputers appear regularly, and a simple alternative to assembly

language can speed the **development** of applications. This shell provides a **processor** independent core and only needs to be matched with a **processor** specific library to provide a **compiler** that accepts Forth input and **generates** absolute **machine code** . The minimum **processor** specific library is derived from the minimal set of primitive words in eForth. In eForth, all other words are derived from these primitives; these same derivations can be used here. You can, of course, define other words as primitives, in the interests of speed, but it is not required that you do so. The **compiler** has been designed to support chips with different word lengths and different architectures; it only expects that the target **processor** **executes** a series of **instructions** taken from some type of ROM and has some RAM in which to **keep** variables and stacks. (0 Refs)

Subfile: C

Descriptors: FORTH; FORTH listings; optimising **compilers** ; software tools; storage management

Identifiers: Nanocomputer Optimizing Target **Compiler** ; **processor** -independent core; **compiler** shell; modern nanocomputer programming; small single chip **processors** ; integrated RAM; assembly language; **processor** independent core; **processor** specific library; Forth input; absolute **machine code** ; minimum **processor** specific library; primitive words; word lengths; target **processor** ; stacks

Class Codes: C6150C (Compilers, interpreters and other processors); C6115 (Programming support); C6140D (High level languages); C6120 (File organisation)



**25/5/37 (Item 10 from file: 2)**

DIALOG(R)File 2:INSPEC



03920598 INSPEC Abstract Number: C91048015

**Title: Poor man's CASE (FDE,** development **environment for VMS)**

Author(s): Weyel, D.

Journal: VAX Professional vol.13, no.2 p.24-7

Publication Date: April 1991 Country of Publication: USA

CODEN: VAXPEN ISSN: 8750-9628

Language: English Document Type: Journal Paper (JP)

Treatment: Practical (P)

Abstract: The FORTRAN **Development** Environment (FDE) is a small menu-driven utility. The FDE **uses** existing VMS facilities to increase programmer productivity on small to medium-sized programming projects. All routines are coded in DCL and **use** ASCII escape sequences for screen management. Global symbols in key command lines provide the flexibility to reconstruct a different **development** environment without leaving the main menu. The FDE **employs** the VMS Librarian as a source and **object code** organizer. A separate FDE command extracts text files from the default text library. Another command automatically replaces object modules in the object library if the **source code** compiled without errors. The Librarian also maintains a limited history of module changes. The FDE remembers which module you are working on, and whether you want to **create** a **compiler** listing. It will automatically compile and link with the /DEBUG option when requested. These **memory** features sharply reduce the time spent issuing repetitive, tedious commands. A VMS shell (i.e., spawn) is included, as is an option to switch editors. However, the FDE's raw simplicity is my favorite feature. I have called it the FORTRAN **Development** Environment, but you might call it the C, or MACRO, or COBOL **Development** Environment. (3 Refs)

Subfile: C

Descriptors: DEC **computers** ; job control language listings; programming environments; public domain software; utility programs

Identifiers: FORTRAN **Development** Environment; menu-driven utility; VMS facilities; VMS Librarian

Class Codes: C6115 (Programming support); C6150E (General utility programs)

**25/5/38 (Item 11 from file: 2)**
DIALOG(R)File 2:INSPEC


03843783 INSPEC Abstract Number: C91022773
**Title: The performance of linear algebra subprograms on the Siemens S series**
Author(s): Schnepf, E.
Author Affiliation: Siemens AG, Munich, West Germany
Conference Title: CONPAR '90-VAPP IV. Joint International Conference on Vector and Parallel Processing. Proceedings p.559-68
Editor(s): Burkhart, H.
Publisher: Springer-Verlag, Berlin, Germany
Publication Date: 1990 Country of Publication: West Germany xi+900 pp.
ISBN: 3 540 53065 7
Conference Date: 10-13 Sept. 1990 Conference Location: Zurich, Switzerland
Language: English Document Type: Conference Paper (PA)
Treatment: Theoretical (T)
Abstract: The architecture of the Siemens S series vector **processors** is presented. The vector unit with its vector pipelines is described in detail. An outstanding feature of the Siemens vector **processors** is the ability to dynamically reconfigurate the vector registers. This allows an optimal adaptation of a given application program to the VP hardware. The **compiler** performs a global vector register optimization to select suitable vector register configurations for all loops of a program and to reduce the number of loads/ **store** operations by combining loops. The optimal **use** of the vector registers and pipelines is supported by a powerful set of vector **instructions**, including combined arithmetical operations and complicated data access methods. The performance of linear algebra subprograms on the S series vector **processors** is explained for single loops, the matrix-vector multiplication and linear equation solvers. The performance that can be obtained on a specific S series model strongly depends on the selected algorithm. The pipeline **utilization** for several algorithms is described. These examples demonstrate how the autovectorizing **compiler** FORTRAN77/VP **generates** an **object code** that **utilizes** the pipelines of the Siemens vector **processors** in the best way. (7 Refs)
Subfile: C
Descriptors: linear algebra; parallel architectures; performance evaluation
Identifiers: performance; linear algebra subprograms; Siemens S series; vector **processors**; vector pipelines; optimal adaptation; single loops; matrix-vector multiplication; linear equation solvers
Class Codes: C5470 (Performance evaluation and testing); C4140 (Linear algebra)

**25/5/39 (Item 12 from file: 2)**
DIALOG(R)File 2:INSPEC


03749762 INSPEC Abstract Number: C90067041
**Title: A reconfigurable** microprocessor **teaching tool**
Author(s): Diab, H.; Demashkieh, I.
Author Affiliation: Dept. of Electr. Eng., American Univ. of Beirut, New York, NY, USA
Journal: IEE Proceedings A (Physical Science, Measurement and Instrumentation, Management and Education) vol.137, no.5 p.287-92
Publication Date: Sept. 1990 Country of Publication: UK
CODEN: IPPRDI ISSN: 0143-702X
U.S. Copyright Clearance Center Code: 0143-702X/90/$3.00+0.00
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: The authors describe a reconfigurable **microprocessor** teaching tool (RMTT) package for **use** in a **microprocessor** systems course. It **uses** the Z80 CPU, or a user-defined 8-bit CPU that is similar to the

internal architecture of the Z80, as the basis for describing how an 8-bit CPU functions internally and as the master of a **microcomputer** system. The package, which consists of an **assembler** , a **control unit** emulator and a graphics simulator, is used as a powerful teaching tool that enables the student to learn about the internal architecture of a **microprocessor** , as **applied** to an 8-bit CPU and a user-defined **instruction** set with a step-by-step graphics animation of the **instruction execution** and timing. The package allows the user to **execute** a program step by step and to test the operation of the internal registers, buses and **memory** contents at every clock edge. This will help the student to understand exactly how the hardware works, for any user-defined **instruction** set, at the **instruction** cycle, machine cycle or clock cycle level. It also simulates read/write cycles from/ **memory** and input/output devices. It allows the user to write and debug programs at the assembly language or **machine - code** level. The low cost and high reliability of the package makes it particularly attractive to teaching establishments as an efficient teaching aid in the introductory course on **microprocessor** systems. The package is menu-driven, interactive, flexible and user-friendly. (12 Refs)

Subfile: C

Descriptors: **computer** aided **instruction** ; **computer** science education ; **development** systems; **microcomputer** applications; software packages; teaching; virtual machines

Identifiers: menu driven interactive package; CAI; reconfigurable **microprocessor** teaching tool; **microprocessor** systems course; Z80 CPU; **control unit** emulator; graphics simulator; internal architecture; user-defined **instruction** set; **instruction execution** ; 8 bit

Class Codes: C0220 (Education and training); C7810C (Computer-aided instruction); C7430 (Computer engineering)

Numerical Indexing: word length 8.0E+00 bit

**25/5/40 (Item 13 from file: 2)**

DIALOG(R)File 2:INSPEC



03623368 INSPEC Abstract Number: B90031270, C90034898

**Title: Managing large software systems**

Author(s): Johns, K.

Journal: Telesis vol.16, no.3 p.50-8

Publication Date: 1989 Country of Publication: Canada

CODEN: TLSSAO ISSN: 0040-2710

Language: English Document Type: Journal Paper (JP)

Treatment: Applications (A); Practical (P)

Abstract: Discusses the Product **Development** Environment (PDE), which provides an integrated set of design and tracking tools for the **development** and delivery of high-quality software. The PDE provides a set of **computerised** tools and databases to link various **development** activities and to give up-to-date product status reports. PDE provides software engineers at all BNR sites with a common set of tools, including: **compilers** to convert **instructions** written in a source language into **machine code** that can be read and acted upon by the DMS hardware; the Product Library System, which **stores** and manages the product's source files; and the Product Tracking System, which tracks the progress of individual features, the **development** and **execution** of test cases, and the resolution of internal and external problem reports. (0 Refs)

Subfile: B C

Descriptors: electronic switching systems; software tools; telecommunications computing

Identifiers: software systems management; design tools; ESS; Product **Development** Environment; tracking tools; **computerised** tools; databases; **compilers** ; source language; **machine code** ; DMS hardware; Product Library System; source files; Product Tracking System

Class Codes: B6230B (Electronic telephone exchanges); C6115 (Programming support); C7410F (Communications)

**25/5/41 (Item 14 from file: 2)**

DIALOG(R)File 2:INSPEC


03347330 INSPEC Abstract Number: C89029460
**Title: The language pyramid**
Author(s): von Kaenel, P.A.
Author Affiliation: Dept. of Math. & Comput. Sci., Skidmore Coll., Saratoga Springs, NY, USA
Journal: Journal of Computing in Small Colleges vol.4, no.2 p. 211-19
Publication Date: Nov. 1988 Country of Publication: USA
Conference Title: Fourth Annual Eastern Small College Computing Conference
Conference Date: 21-22 Oct. 1988 Conference Location: USA
Language: English Document Type: Conference Paper (PA); Journal Paper (JP)
Treatment: Practical (P)
Abstract: A multilevel **computer** can be viewed as a sequence of virtual machines, each with a different machine language. At each level, programs are **executed** by interpreters running at a lower level or compiled into another language corresponding to a lower level. To provide a simple environment that is rich enough for multiple level work, the author has introduced a software simulator of a digital watch that is programmed at the microprogram level and includes random access **memory** for **storing** machine level **instructions** and data. The watch provides an ideal environment for studying multilevel machines. The design considerations and examples of the microprogram, machine, assembly, and operating system levels that run on the watch are presented. The watch's simplicity results in small, clean languages, ideally suited for class work. In addition, higher level, Pascal-like languages can be **developed** . **Compilers** for this level as well as **assemblers** must be written to **produce** machine language files that are used as input to the watch program. (2 Refs)
Subfile: C
Descriptors: **computer** aided **instruction** ; **computer** science education ; digital simulation; operating systems ( **computers** ); program **assemblers** ; program **compilers** ; program interpreters; programming languages; virtual machines
Identifiers: program compilation; language levels; **compilers** ; microprogramming; **machine code** ; assembly language; high level languages ; multilevel **computer** ; virtual machines; machine language; interpreters; software simulator; digital watch; random access **memory** ; machine level **instructions** ; design considerations; operating system; Pascal-like languages; **assemblers**
Class Codes: C7430 (Computer engineering); C6140 (Programming languages ); C0220 (Education and training); C6150C (Compilers, interpreters and other processors); C6150J (Operating systems); C7810C (Computer-aided instruction)

**25/5/42 (Item 15 from file: 2)**
DIALOG(R)File 2:INSPEC


02704138 INSPEC Abstract Number: C86038970
**Title: A workstation for the** development **of iAPX186 based multiprocessor systems**
Author(s): Albertengo, G.
Author Affiliation: Dipartimento di Elettronica, Politecnico di Torino, Italy
Conference Title: Mini and Microcomputers and their Applications: MIMI '84 Bari. Proceedings of the ISMM International Symposium p.114-17
Editor(s): Mastronardi, G.
Publisher: Acta Press, Anaheim, CA, USA
Publication Date: 1984 Country of Publication: USA 289 pp.
ISBN: 0 88986 058 0
Conference Date: 5-8 June 1984 Conference Location: Bari, Italy
Language: English Document Type: Conference Paper (PA)

Treatment: Practical (P)
Abstract: A versatile workstation for multiprocessor system **development** was built **using** a Digital Equipment Corporation (DEC) PDP11/03 minicomputer. The station monitors up to seven iAPX186 **microprocessors** by means of a simple interface circuit and of a supervisory program resident in the targets. The station control program offers some simple-to- **use** and powerful tools for **microprocessor** program **development** . One can load and verify the program in the target; stop and resume its **execution** ; insert up to sixteen software breakpoints; read and modify all **memory** and peripheral existing locations and all CPU registers. Furthermore, running the program in the step-by-step mode, one can trace it. In the actual configuration, the station is connected to a DEC VAX11/780 via a 4800 baud serial line. On the mainframe a MACRO32 based cross- **assembler** performs **source code** assembly, and a **linker generates** the **machine code** for the target **microprocessor** . A communication utility performs file transfer between the station and the mainframe. (0 Refs)
Subfile: C
Descriptors: multiprocessing systems; workstations
Identifiers: workstation; iAPX186 based multiprocessor systems; Digital Equipment Corporation; PDP11/03 minicomputer; interface circuit; supervisory program; **microprocessor** program **development** ; DEC VAX11/780; MACRO32 based cross- **assembler** ; communication utility; file transfer
Class Codes: C5430 (Microcomputers); C5440 (Multiprocessor systems and techniques)

**25/5/43 (Item 16 from file: 2)**
DIALOG(R)File 2:INSPEC


02660367 INSPEC Abstract Number: C86031465
**Title: Converting SPICE to vector code**
Author(s): Greer, B.
Author Affiliation: Floating Point Syst. Inc., Beaverton, OR, USA
Journal: VLSI Design , vol.7, no.1, p.30-2
Publication Date: Jan. 1986 Country of Publication: USA
CODEN: VDESDP ISSN: 0279-2834
Language: English Document Type: Journal Paper (JP)
Treatment: General, Review (G)
Abstract: In reformatting SPICE for **execution** on Floating Point Systems' scientific **computers** , the author's purpose was to convert the scalar code of the model evaluation routines into vector code. This procedure introduces DO loops, for which an optimizing FORTRAN **compiler** can **generate** efficient **machine code** . These DO loops do not process data one transistor at a time; instead, they evaluate a particular function for all the relevant transistors, before beginning the next function for all transistors. However, certain branches in the scalar code depend on the value of particular parameters associated with the device currently being evaluated, While these branches could be dealt with within a DO loop through the **use** of IF **statements** , IF **statements** within the loop impact efficiency. Therefore, a way had to be devised which would perform the equivalent of an IF **statement** for each device. A second problem arose from SPICE's linked-list approach for data storage, which results in an essentially random ordering of devices within **memory** . To get data about a particular device requires going through the complete linked lists for all of the devices preceding the device of interest. To resolve this dilemma, the data must be reordered such that the distance between tables is fixed, with all integer tables grouped together and all real tables grouped together. (0 Refs)
Subfile: C
Descriptors: circuit CAD; computational complexity
Identifiers: SPICE conversion to vector code; data reordering; reformatting SPICE; Floating Point Systems' scientific **computers** ; DO loops; optimizing FORTRAN **compiler** ; IF **statements** ; linked-list approach for data storage
Class Codes: C4240 (Programming and algorithm theory); C7410D ( Electronic engineering)

**25/5/44 (Item 17 from file: 2)**
DIALOG(R)File 2:INSPEC


02423849 INSPEC Abstract Number: C85020350
**Title: A study of code generation for the** language processor **generator MYLANG**
Author(s): Ohba, E.; Yamanoue, T.; Anzai, H.
Author Affiliation: Kyushu Inst. of Technol., Tabata, Japan
Journal: Bulletin of the Kyushu Institute of Technology (Science and Technology) no.48 p.47-55
Publication Date: March 1984 Country of Publication: Japan
CODEN: KKDKAN ISSN: 0453-0357
Language: Japanese Document Type: Journal Paper (JP)
Treatment: Theoretical (T)
Abstract: The authors present a study of code generation and its optimisation concerned with functional extension of the **language processor** generator MYLANG. Two topics are dealt with. One is a problem of register assignment to an arithmetic expression. The other is a problem of code generation from a labelled automaton which is constructed by the MYLANG system. For the optimal assignment of registers to a given arithmetic expression, the Sethi and Ullman's algorithm is used. This algorithm decreases load and **store instructions** and shortens the **execution** time of an **object code** transformed from the arithmetic expression. It is shown how to **generate** the **object code** from the labeled automaton. (9 Refs)
Subfile: C
Descriptors: program **processors**
Identifiers: code generation; **language processor** generator; MYLANG; optimisation; register assignment; code generation; labelled automaton; Sethi and Ullman's algorithm
Class Codes: C6150C (Compilers, interpreters and other processors)

**25/5/45 (Item 18 from file: 2)**
DIALOG(R)File 2:INSPEC


02125809 INSPEC Abstract Number: C83039299
**Title: Extensions to static scoping**
Author(s): Cormack, G.V.
Author Affiliation: School of Computer Sci., McGill Univ., Montreal, Que., Canada
Journal: SIGPLAN Notices vol.18, no.6 p.187-91
Publication Date: June 1983 Country of Publication: USA
CODEN: SINODQ ISSN: 0362-1340
U.S. Copyright Clearance Center Code: 0362-1340/83/006/0187$00.75
Conference Title: Proceedings of the SIGPLAN '83 Symposium on Programming Language Issues in Software Systems
Conference Sponsor: ACM
Conference Date: 27-29 June 1983 Conference Location: San Francisco, CA, USA
Language: English Document Type: Conference Paper (PA); Journal Paper (JP)
Treatment: Practical (P)
Abstract: Scope rules are presented as they appear in the language L, which is currently under development at McGill. These rules permit a programmer to specify explicitly the duration and visibility of all objects. Such specification can be used to declare variables or to create data structures which persist from one invocation of a block to the next. In addition, the data may be caused to persist from one run of the program to the next, obviating **temporary files** which are outside the scope and control of the programming language. The visibility of an object may also be specified to be larger than the block that contains its **declaration**. This facility allows the programmer to export some objects, yet **keep**

others private. Finally, a facility for call-site visibility is presented that provides some of the expressive power of dynamic scoping and macros without their inherent type insecurity. (8 Refs)
Subfile: C
Descriptors: high level languages
Identifiers: ALGOL; high level languages; static scoping; language L; visibility; objects; data structures; **temporary files** ; dynamic scoping; macros; type insecurity
Class Codes: C6140D (High level languages)

**25/5/46 (Item 19 from file: 2)**
DIALOG(R)File 2:INSPEC
(c) 2004 Institution of Electrical Engineers. All rts. reserv.

01908270 INSPEC Abstract Number: C82036202
**Title: SDS: editor-monitor-** assembler **for Color** Computer
Author(s): Puckett, D.
Journal: InfoWorld vol.4, no.1 p.30-2
Publication Date: 11 Jan. 1982 Country of Publication: USA
CODEN: INWODU ISSN: 0199-6649
Language: English Document Type: Journal Paper (JP)
Treatment: Applications (A)
Abstract: The Software **Development** System is a new programming tool for the Radio Shack Color **Computer** . It comes in a 'ROM-pack' that plugs into the side of the machine. The SDS package contains three distinct programs that make it easy to write short assembly-language programs: for typing an editor **assembly - language source code** into **memory** , an **assembler** that converts the mnemonic symbols of this **source code** into the actual **machine code executed** by the Color **Computer** 's 6809 **microprocessor** , and a monitor program called ABUG that oversees machine-language programs while they are running. (0 Refs)
Subfile: C
Descriptors: personal computing; program **assemblers** ; program debugging; text editing
Identifiers: Software **Development** System; Radio Shack Color **Computer** ; editor; **assembler** ; monitor program
Class Codes: C6150C (Compilers, interpreters and other processors); C6150G (Diagnostic, testing, debugging and evaluating systems); C7830 ( Home computing)

**25/5/47 (Item 20 from file: 2)**
DIALOG(R)File 2:INSPEC
(c) 2004 Institution of Electrical Engineers. All rts. reserv.

01672058 INSPEC Abstract Number: C81015566
**Title: An interactive simulator for the KIM-1** microcomputer
Author(s): Nagata, W.M.; Miller, D.S.
Author Affiliation: Pacific Northwest Bell, Seattle, WA, USA
Journal: Simulation vol.36, no.1 p.21-33
Publication Date: Jan. 1981 Country of Publication: USA
CODEN: SIMUA2 ISSN: 0037-5497
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: The KIM-1 simulator is a **development** tool for the MOS Technology 6502 **microprocessor** and the KIM-1 **microcomputer** . It is written in the C programming language and runs under the UNIX operating system on a DEC PDP-11/60. The input to the simulator is **object code produced** by a cross- **assembler** . The simulator helps in debugging 6502 programs by providing the user with the ability to interactively load and display **memory** and registers, dynamically set and clear breakpoints, handle interrupts, and trace program **execution** . The simulator is user-oriented; it supplies prompts, English diagnostics, and **instructions** on usage. Its **execution** speed and core requirements allow truly interactive debugging. (18 Refs)
Subfile: C

Descriptors: interactive systems; **microcomputers** ; program debugging; virtual machines
Identifiers: interactive simulator; KIM-1 **microcomputer** ; **development** tool; MOS Technology 6502 **microprocessor** ; C programming language; UNIX operating system; DEC PDP-11/60; debugging; **memory** ; registers; breakpoints; interrupts; program **execution**
Class Codes: C5250 (Microcomputer techniques); C7430 (Computer engineering)

**25/5/48 (Item 21 from file: 2)**
DIALOG(R)File 2:INSPEC


01224003 INSPEC Abstract Number: B78034813, C78019922
**Title: SPC-1 support software**
Journal: Telecommunications vol.27, no.1 p.77-80
Publication Date: June 1977 Country of Publication: India
CODEN: TCMSAX ISSN: 0497-1388
Language: English Document Type: Journal Paper (JP)
Treatment: Applications (A); Practical (P)
Abstract: SPC-1 programs are written in a Macro-Assembly language which **uses** symbolic **instructions** as well as macros. Support software, comprising of a **compiler** - **assembler** and a loader has been **developed** to convert the source program, written in the above language, into an object program for loading into the program **store** of the SPC-1 switching **processor** . To test the logic of the SPC-1 system software, in advance of the availability of the SPC-1 switching **processor** , an Interpretive Simulator has also been **developed** . At present the **Compiler** - **Assembler** Loader as well as the Simulator have been written in a version of Algol combined with **machine** - **code** language of Elliot 803B **Computer** , available at the Telecommunication Research Centre of the Indian Posts & Telegraphs Department. They will also be written in Fortran IV and PL-1 to make them machine-transportable. (0 Refs)
Subfile: B C
Descriptors: electronic switching systems; macros; simulation
Identifiers: SPC-1 support software; macros; interpretive simulator
Class Codes: B6230B (Electronic telephone exchanges); C3370C (Telephony); C7410F (Communications)

**25/5/49 (Item 22 from file: 2)**
DIALOG(R)File 2:INSPEC


01060438 INSPEC Abstract Number: C77015850
**Title: A high level language for IMP-16**
Journal: Compute vol.2, no.8 p.6 pp.
Publication Date: Aug. 1976 Country of Publication: USA
ISSN: 0194-357X
Language: English Document Type: Journal Paper (JP)
Treatment: Practical (P)
Abstract: Discusses SM/PL (Smart or Simple Programming Language), a high level language for the IMP-16 **Microprocessor** . SM/PL requires 16K of read/write **memory** , and can be used with the IMP-16 Disc Operating System. It accepts source **statements** **generated** by **using** EDIT16 and will **produce** a listing of the assembly code **generated** or an object program. The **compiler** includes a number of features such as access to the IMP-16 machine facilities, linkage to assembly language routines and various **compiler** control **statements** for source listing, interlisting of assembly and **object** **code** , and symbol table dumps. A list of some of the language features is shown. (0 Refs)
Subfile: C
Descriptors: **microcomputers** ; programming languages
Identifiers: high level language; IMP-16 **Microprocessor** ; IMP-16 Disc Operating System; language features; 16K RAM requirement; SM/PL language
Class Codes: C5280 (Other digital techniques); C6140D (High level

languages)

25/5/50 (Item 23 from file: 2)
DIALOG(R)File 2:INSPEC


00301503 INSPEC Abstract Number: C71019914
**Title: CONFORM-a** compiler **for process** control computer **systems**
Author(s): Arita, F.; Sudo, M.; Sekimoto, S.; Iharada, K.
Journal: Mitsubishi Denki Giho vol.45, no.2 p.213-20
Publication Date: Feb. 1971 Country of Publication: Japan
CODEN: MTDNAF ISSN: 0369-2302
Language: Japanese Document Type: Journal Paper (JP)
Treatment: New Developments. (N)
Abstract: CONFORM for the **computer** MELCOM-350/30 is a language designed for the ease of user programming in the industrial **computer** control. It is a FORTRAN extension for **use** in the process control, incorporating the functions such as bit-string operations, **executive** interface calls, process input/output calls, and external **memory** read/write operations. The extensions are made at a syntactical level rather than by simple attachment of subroutine calls, so that the **compiler generates** efficient **object codes** . **Assembler instructions** can be inserted at any position of CONFORM program. Problem oriented programming systems, such as fill-in-blank programming languages, are to be **developed** on this language as their basis. This article describes the design philosophy, outlines of language specification, and the organization of the **compiler** and object programs.
Subfile: C
Descriptors: control engineering applications of **computers** ; process control; program **compilers**
Identifiers: bit string operations; FORTRAN; process control; **executive** interface calls; process input/output calls; external **memory** read/write operations; syntactical level; **compiler** ; efficient **object codes** ; programming languages; design philosophy; language specification; organization; **compiler** and object programs
Class Codes: C6150C (Compilers, interpreters and other processors); C7420 (Control engineering)

25/5/51 (Item 1 from file: 94)
DIALOG(R)File 94:JICST-EPlus


04681823 JICST ACCESSION NUMBER: 00A0888864 FILE SEGMENT: JICST-E
**Cogeneration of an Embedded** Microprocessor **and Its** Object Code **to Minimize** Memory **Consumption.**
NAKANO TAKESHI (1); NAKANISHI TSUNEO (1); FUKUDA AKIRA (1)
(1) Advanced Inst. Sci. and Technol., Nara
Joho Shori Gakkai Kenkyu Hokoku, 2000, VOL.2000,NO.74(ARC-139), PAGE.139-144, FIG.6, TBL.1, REF.12
JOURNAL NUMBER: Z0031BAO ISSN NO: 0919-6072
UNIVERSAL DECIMAL CLASSIFICATION: 681.32 681.3.068
LANGUAGE: Japanese COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: In this paper we present a sketch of PinkPanther, a cogenerater which **generates** application-specific **microprocessor** and its **object code** for embedded systems with limited **memory** capacity. PinkPanther builds an **instruction** set automatically to reduce **memory** consumption. We also present preliminary evaluation of the code packing algorithm **employed** by PinkPanther. (author abst.)
DESCRIPTORS: **computer** architecture; **compiler** ; program generator; **instruction** set; **microprocessor** ; customizing; special purpose **processor** ; code generation; benchmark( **computer** ); data compression; performance evaluation

BROADER DESCRIPTORS: **computer** system(architecture); method; **language processor** ; system program; **computer** program; software; arithmetic **processor** ; hardware; modification; data processing; information processing; treatment; evaluation
CLASSIFICATION CODE(S): JC02010O; JD03040F

**25/5/52 (Item 2 from file: 94)**
DIALOG(R)File 94:JICST-EPlus


04453752 JICST ACCESSION NUMBER: 00A0011252 FILE SEGMENT: JICST-E
**VLSI Design and CAD Algorithms. A** Memory **Power Optimization Technique for Application Specific Embedded Systems.**
ISHIHARA T (1); YASUURA H (1)
(1) Kyushu Univ., Kasuga-shi, Jpn
IEICE Trans Fundam Electron Commun Comput Sci(Inst Electron Inf Commun Eng) , 1999, VOL.E82-A,NO.11, PAGE.2366-2374, FIG.14, TBL.4, REF.10
JOURNAL NUMBER: F0699CAT ISSN NO: 0916-8508
UNIVERSAL DECIMAL CLASSIFICATION: 681.325/.326.009.16 621.3.049.77 681.32.07
LANGUAGE: English COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: In this paper, a novel application specific power optimization technique **utilizing** small **instruction** ROM which is placed between an **instruction cache** or a main program **memory** and CPU core is proposed. Our optimization technique targets embedded systems which assume the following: (i) **instruction** memories are organized by two on-chip memories, a main program **memory** and a subprogram **memory** , (ii) these two memories can be independently powered-up or powered-down by a special **instruction** of a core **processor** , and (iii) a **compiler** optimizes an allocation of **object code** into these two memories so as to minimize average of read energy consumption. In many application programs, only a few basic blocks are frequently **executed** . Therefore, allocating these frequently **executed** basic blocks into low power subprogram **memory** leads significant energy reduction. Our experiments with actual ROM (Read Only **Memory** ) modules **created** with 0.5.MU.m CMOS process technology, and MPEG2 codec program demonstrate significant energy reductions up to more than 50% at best case over the previous approach that **applies** only divided bit and word lines structure. (author abst.)
DESCRIPTORS: interposition; special purpose **processor** ; ASIC; data reading ; **cache memory** ; ROM; consumed electric power; **compiler** ; hardware design; software design; **computer** system **development** ; optimization method; storage system; storage management
IDENTIFIERS: ASIP; **memory** access
BROADER DESCRIPTORS: insertion; hardware; integrated circuit; micro circuit ; data processing; information processing; treatment; **memory** ( **computer** ); equipment; electric power; **language processor** ; system program; **computer** program; software; design; **development** ; method; management
CLASSIFICATION CODE(S): JC04030Y; NC03161C; JC02040V

**25/5/53 (Item 3 from file: 94)**
DIALOG(R)File 94:JICST-EPlus


04453748 JICST ACCESSION NUMBER: 00A0011248 FILE SEGMENT: JICST-E
**VLSI Design and CAD Algorithms. A Hardware/Software Cosynthesis System for Digital Signal** Processor **Cores.**
TOGAWA N (1); YANAGISAWA M (1); OHTSUKI T (1)
(1) Waseda Univ., Tokyo, Jpn
IEICE Trans Fundam Electron Commun Comput Sci(Inst Electron Inf Commun Eng) , 1999, VOL.E82-A,NO.11, PAGE.2325-2337, FIG.6, TBL.8, REF.36

JOURNAL NUMBER: F0699CAT ISSN NO: 0916-8508
UNIVERSAL DECIMAL CLASSIFICATION: 681.3.02.001 681.325/.326.009.16
LANGUAGE: English COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: This paper proposes a hardware/software cosynthesis system for digital signal **processor** cores and a hardware/software partitioning algorithm which is one of the key issues for the system. The target **processor** has a VLIW-type core which can be composed of a **processor** kernel, multiple data **memory** buses (X-bus and Y-bus), hardware loop units, addressing units, and multiple functional units. The **processor** kernel includes five pipeline stages (RISC-type kernel) or three pipeline stages (DSP-type kernel). Given an application program written in the C language and a set of application data, the system synthesizes a **processor** core by selecting an appropriate kernel (RISC-type or DSP-type kernel) and required hardware units according to the application program/data and the hardware costs. The system also **generates** the **object code** for the application program and a software environment ( **compiler** and simulator) for the **processor** core. The experimental results demonstrate that the system synthesizes **processor** cores effectively according to the features of an application program and the synthesized **processor** cores **execute** most application programs with the minimum number of clock cycles compared with several existing **processors** . (author abst.)
DESCRIPTORS: DSP( **processor** ); hardware design; software design; concurrency; **computer** architecture; **computer** system **development** ; RISC machine; C language; code generation; application program; particle size(ratio
IDENTIFIERS: very long **instruction** word architecture
BROADER DESCRIPTORS: special purpose **processor** ; hardware; **microprocessor** ; arithmetic **processor** ; design; property; **computer** system(architecture); method; **development** ; digital **computer** ; **computer** ; high level language; programming language; formal language; language; **compiler** ; **language processor** ; system program; **computer** program; software; degree
CLASSIFICATION CODE(S): JD02010R; JC04030Y

**25/5/54 (Item 4 from file: 94)**
DIALOG(R)File 94:JICST-EPlus


04010743 JICST ACCESSION NUMBER: 99A0411787 FILE SEGMENT: JICST-E
**A Power Optimization Technique for Application Specific** Memory **Designs.**
ISHIHARA TOORU (1); YASUURA HIROTO (1)
(1) Kyushu Univ., Grad. Sch.
Denshi Joho Tsushin Gakkai Gijutsu Kenkyu Hokoku(IEIC Technical Report (Institute of Electronics, Information and Communication Enginners), 1999, VOL.98,NO.626(ICD98 287-295), PAGE.1-8, FIG.13, TBL.4, REF.8
JOURNAL NUMBER: S0532BBG
UNIVERSAL DECIMAL CLASSIFICATION: 621.382.2/.3.049.77 681.326.32
LANGUAGE: Japanese COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: In this paper, a novel application specific power optimization technique **utilizing** well known microprogramming technique is proposed. Our proposed optimization technique targets systems which assume the following: (i) **instruction memory** is organized by two on-chip memories, a main program **memory** and a microprogram **memory** , (ii) these two memories can be independently powered-up or powered-down by a special **instruction** of a core **processor** , and (iii) a **compiler** can optimize an allocation of **object code** into these two memories so as to minimize average read energy consumption. In many application programs, only a few basic blocks are frequently **executed** . Therefore, packing these frequently **executed** basic blocks into low power

microprogram **memory** leads significant energy reduction. Our experiments with actual ROM(Read Only **Memory** ) designs **created** with 0.5.MU.m CMOS process technology, and MPEG2 codec program demonstrate significant energy reductions up to more than 80% at beat case over the previous approach that **applies** only divided bit lines structure or divided word lines structure. (author abst.)
DESCRIPTORS: ROM; SRAM; consumed electric power; optimization method; turn around time( **computer** ); microprogramming; programming system; LSI; **computer** architecture
IDENTIFIERS: microprogram; architerture
BROADER DESCRIPTORS: **memory** ( **computer** ); equipment; RAM; static **memory** ; electric power; **computer** processing characteristic; **computer** characteristic; characteristic; time; **computer** programming; system; integrated circuit; micro circuit; **computer** system(architecture); method
CLASSIFICATION CODE(S): NC03162T; JD03060B

**25/5/55 (Item 5 from file: 94)**
DIALOG(R)File 94:JICST-EPlus


02004055 JICST ACCESSION NUMBER: 94A0137329 FILE SEGMENT: JICST-E
Compiler **Generation for** Computer **Architecture Evaluation.**
TOMIYAMA HIROYUKI (1); AKABOSHI HIROKI (2); YASUURA HIROTO (2)
(1) Kyushu Univ., Faculty of Engineering; (2) Kyushu Univ.
Joho Shori Gakkai Kenkyu Hokoku, 1993, VOL.93,NO.111(ARC-103 DA-69), PAGE.143-150, FIG.4, TBL.7, REF.10
JOURNAL NUMBER: Z0031BAO ISSN NO: 0919-6072
UNIVERSAL DECIMAL CLASSIFICATION: 681.3.068 681.32
LANGUAGE: Japanese COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: **Compiler** generation from a **computer** architecture description in an HDL(Hardware Description Language) makes a designer evaluate his architecture easily. It also provides a software **development** environment. We are **implementing** a prototype of a **compiler** generator **using** the Tree Rewriting algorithm. The **compiler** generator **produces** a **compiler** for a designed architecture with various kinds of **instructions** . Experimental results show that **compilers** **generated** by the **compiler** generator **generate** near-optimal **object** **code** while there is room to improve the register allocation method. (author abst.)
DESCRIPTORS: **compiler** ; **computer** architecture; program generator; rewriting rule; tree(graph); **instruction** set; code generation; **memory** allocation; schematic design; performance evaluation; prototyping( **computer** )
BROADER DESCRIPTORS: **language** **processor** ; system program; **computer** program; software; **computer** system(architecture); method; rule; subgraph; graph; storage management; management; design; evaluation; **computer** system **development** ; **development**
CLASSIFICATION CODE(S): JD03040F; JC02010O

**25/5/56 (Item 6 from file: 94)**
DIALOG(R)File 94:JICST-EPlus


00403949 JICST ACCESSION NUMBER: 87A0206083 FILE SEGMENT: JICST-E
**The optimization of C-Prolog** compiler .
KOBAYASHI SHIGERU (1); MATSUMOTO NORIYUKI (1); OCHIAI MASAO (1); HON'IDEN SHIN'ICHI (1)
(1) Toshiba Fuchukojo
Joho Shori Gakkai Kenkyu Hokoku, 1987, VOL.87,NO.9(PL-10), PAGE.10.4.1-10.4.8, FIG.8, REF.4
JOURNAL NUMBER: Z0031BAO ISSN NO: 0919-6072

UNIVERSAL DECIMAL CLASSIFICATION: 681.3.068
LANGUAGE: Japanese COUNTRY OF PUBLICATION: Japan
DOCUMENT TYPE: Journal
ARTICLE TYPE: Original paper
MEDIA TYPE: Printed Publication
ABSTRACT: We deveroped a C-Prolog **compiler** on G8050, a super-mini **computer** . This **compiler** works as a subsystem of the C-Prolog interpreter. It **generates** object programs on main **memory** from source files. It **uses** **instruction** set suggested by D.H. Warren as the base of intermediate code. To optimize the efficiency of the **object** **code** , we broke down and added new **instructions** to Warren's **instruction** set. Not only clause-local optimizations, we also examined inter-clause or inter-predicate wided optimizations such as delaying of **saving** argument registers or adopting global variables.(author abst.)
DESCRIPTORS: **compiler** ; Prolog; software design; optimization; interpreter ; code generation; symbol processing; predicative logic
BROADER DESCRIPTORS: **language** **processor** ; system program; **computer** program; software; high level language; programming language; formal language; language; design; modification; information processing; treatment; logic
CLASSIFICATION CODE(S): JD03040F

**25/5/57 (Item 1 from file: 95)**
DIALOG(R)File 95:TEME-Technology & Management


01012143 E96087142031
**Code generation and optimization techniques for embedded digtital signal** processors
(Ueber Code-Generierung und Optimierungstechniken bei eingebetteten digitalen Signalprozessoren)
Liao, S; Devados, S; Keutzer, K; Tjiang, S; Wang, A; Aranjo, G; Sudarsanam, A; ua
MIT Cambridge, USA; Synopsis, Mountain View, USA; Princeton Univ., USA; u.a.
Hardware/Software Co-Design, Proc. of the NATO Adv. Study Inst., Tremezzo, I, Jun 19-30, 19951996
Document type: Conference paper Language: English
Record type: Abstract
ISBN: 0-7923-3882-0

ABSTRACT:
The authors have explored the programming language **compiler** requirements for embedded **processors** , and present several optimizations that are effective in producing high-performance and dense **object** **code** . The authors believe that final code of the highest possible quality is necessary because the embedded software will eventually become part of the chip and will be **produced** in large volumes. Achieving this code quality requires **employing** the entire suite of classical **compiler** optimization techniques, and also new techniques. In addition, it motivates lifting the typical O(n) or O(n log n) limit on the running time of the algorithms to solve these optimization problems. In this chapter the authors have examined problems that arise from non-homogeneous **instruction** sets, irregular register structure, limited addressing modes, parallel **memory** access, and code compression. Other idiosyncrasies such as zero-overhead loop **instructions** will also need special treatment.

DESCRIPTORS: PROGRAM GENERATORS; AUTOMATIC PROGRAMMING; CHIPS--SEMICONDUCTORS; **COMPUTERISED** SIGNAL PROCESSING; **COMPILERS** ; TRANSLATING PROGRAM; IMPROVEMENT
IDENTIFIERS: Code-Generator; Code-Optimierung

**25/5/58 (Item 2 from file: 95)**
DIALOG(R)File 95:TEME-Technology & Management

00637940 E93013034020
**A parallel incremental architecture for Prolog program** execution
(Eine parallel inkrementelle Architektur fuer die Ausfuehrung von Prolog-Programmen)
Gloria, Ade; Faraboschi, P; Guidetti, E
Univ. of Genoa, I
VLSI for Artificial Intelligence and Neural Networks, Proceedings of the International Workshop, Oxford, GB, September 5-7, 19901991
Document type: Conference paper Language: English
Record type: Abstract
ISBN: 0-306-44029-6

ABSTRACT:
Logic programming is nowadays a subject of large attention, as many researchers believe that it can map AI problems into **machine code** more easily than traditional high-level languages. Besides, the demand of increasing computation power for symbolic processing and the availability of VLSI products with low fabrication cost is giving a strong impulse to the **development** of integrated circuits dedicated to the **execution** of symbolic languages, instead of the adaptation of general purpose **processors** as in recent past. SYMBOL represents an experiment in applying RISC design philosophy and compiling techniques derived from scientific computation to the exploitation of fine-grain parallelism in Prolog compiled programs. Gate-level simulations of a three- **processors** architecture realized with a 2 micron standard cell technology and operating at 30 MHz exhibits 600 KLips average performance on standard Prolog benchmarks and 1.2 Mlips peak.

DESCRIPTORS: **COMPUTERIZED** SIMULATION; CIRCUIT SIMULATION; MODEL STUDY; PROGRAM FLOW; LOGIC PROGRAMMING; ARTIFICIAL INTELLIGENCE; GRAND SCALE INTEGRATION; CHIPS--SEMICONDUCTORS; REDUCED **INSTRUCTION** SET **COMPUTER** ; **MICROELECTRONICS** ; **COMPUTER ARCHITECTURE** ; COMMAND **EXECUTION** ; PERFORMANCE IMPROVEMENT; **COMPILERS** ; PIPELINE PROCESSING; PARALLEL PROGRAMMING; PARALLEL PROCESSING; **IMPLEMENTATION** ; ASSOCIATIVE MEMORIES; **MEMORY** MANAGEMENT; DATA STORAGE; COMMAND STRUCTURE; PROGRAMMING THEORY; **COMPUTER** PROGRAMMING
IDENTIFIERS: Prolog-Ausfuehrung; Systemarchitektur

**25/5/59 (Item 3 from file: 95)**
DIALOG(R)File 95:TEME-Technology & Management


00563019 I92012114927
**Dual-ALU CRISC architecture and its compiling technique**
(Dual-ALU-CRISC-Architektur und seine **Compiler** -Technik)
Hong-Chich Chou; Chung-Ping Chung; Shyi-Chyi Cheng
Inst. of Comput. Sci. & Inf. Eng., Nat. Chiao Tung Univ., Hsinchu, Taiwan
Computers and Electrical Engineering, v17, n4, pp297-312, 1991
Document type: journal article Language: English
Record type: Abstract
ISSN: 0045-7906

ABSTRACT:
As semiconductor technology advances, more devices can be accommodated in a single VLSI chip. The feasibility of putting multifunctional units in a chip is then worth studying. Such an approach, however, will face software and hardware difficulties (and also tradeoffs). CRISC is a 32-bit single-chip VLSI **processor** architecture achieving high performance by means of RISC and multiple functional unit approaches. Dual-ALUs are used to **execute instructions** concurrently for fine-grained parallelism. Up to three **instructions** can be **executed** simultaneously by CRISC. Here, CRISC architecture design considerations and **instruction cache** scheme are investigated. Final microarchitecture and its incorporated software technique to **produce object code** for fine-grained parallel **execution** are described; its upper bound performance is estimated by an

architectural model. A preliminary evaluation of the CRISC is also conducted, showing most satisfying results.

DESCRIPTORS: PARALLEL PROCESSING; GRAND SCALE INTEGRATION; 32 BIT **MICROPROCESSORS** ; CHIPS--SEMICONDUCTORS; **COMPUTER** ARCHITECTURE; SEMICONDUCTOR TECHNOLOGY; **COMPUTER** PERFORMANCE; PROGRAM **INSTRUCTION** ; CIRCUIT DESIGN; CODE CONVERSION; **COMPILERS** ; **MICROPROCESSORS** ; REDUCED **INSTRUCTION** SET **COMPUTER** ; PARALLEL ARCHITECTURES
IDENTIFIERS: DUAL ALU; CRISC; COMPILING; VLSI **PROCESSOR** ARCHITECTURE; MICROARCHITECTURE; SOFTWARE TECHNIQUE; PARALLEL **EXECUTION** ; Dual-ALU; **Compiler**